Universidad
Carlos III de Madrid

Grado en Ingeniería de Sistemas Audiovisuales

TRABAJO FIN DE GRADO

Diseño e integración en Android de un sistema de domótica asistencial basado en reconocimiento de imágenes

Autor: Paloma Diego Velasco

Tutor: Javier Fernández Muñoz

Leganés, Julio de 2014

Universidad
Carlos III de Madrid

Universidad Carlos III de Madrid

Agradecimientos

Lo siento, pero no soy capaz de poner: a la familia y amigos. He tenido la suerte de tener a mucha gente a la que agradecer, así que me doy por vencida en el intento, trataré de no ser muy pesada, prometido.

En primer lugar y como no podría ser de otro modo, gracias a mi padre por tratar de prepararme para la dureza de la vida, a mi madre por tenerme tan "calada" y a mi hermana que es mi polo positivo y sin duda la mejor compañera que puedo tener en mi vida. Mil y una vez gracias a los tres por confiar en mí y hacerme sentir tan afortunada de teneros, porque sin vosotros nada sería posible. También a: Paloma, Juan, Alejandro y María, mis "amarillos" en momentos difíciles, a mi abuela, de la que espero que algún día se me pegue un poco de su inagotable ternura; y a la última incorporación a mi familia, Óscar, gracias por hacerme creer que puedo, cuando lo veo todo negro, y sobre todo por hacerme sentir tan querida cuando me miras.

Para hablar de los amigos, doy gracias a la suerte de haberos encontrado a todos: los de siempre, los Opencorianos, los SDI, las del aquelarre, los de la antigua titulación, los últimos fichajes, los que siguen en mi vida y los que no, porque con vosotros he crecido y sigo haciéndolo cada día. Muy en especial gracias a mis chicas, Amelín y Sara porque siempre será un buen momento para ir con vosotras a cualquier parte; a los compañeros de dolores de cabeza en estos últimos años de carrera: Diego y María (gracias por esos lunes inaguantables y divertidos) y a Bea y Sue, porque el último año ha sido muy duro para nosotras, pero siempre hemos encontrado la manera de reírnos de todo y seguir, siempre seguir.

De estos años en la universidad, no olvidaré muchos profesores, algunos por su incomprensible "mala fe"; pero también a los que nos enseñan con ganas de verdad. Quiero dar las gracias a Iria Estévez, por sus "no te preocupes mujer...ya verás como hoy te sale", que me han llevado a dedicarme hoy a lo que más me gusta que es la programación; y a Ascensión Gallardo, por buscar solución a cómo podía hacer un examen en el momento más difícil de mi vida. Muchas gracias también a mi tutor Javier Fernández, porque se ha preocupado por enseñarme algunas cosas que me han sido y me están siendo muy útiles a la hora de trabajar en un proyecto.

Por último, quiero dar las gracias a mi abuelo, por una relación más que especial. Gracias por enseñarme el significado de palabras como la perseverancia y la honestidad, valores que me acompañarán en toda mi vida. Infinitas gracias por empeñarte en que estudiara porque gracias a aquellas multiplicaciones interminables y a las mil historias, cuentos y realidades, soy feliz con la vida que estoy construyendo.

De verdad y con mucho cariño, gracias a todos.

Universidad
Carlos III de Madrid

Universidad
Carlos III de Madrid

Resumen

Este proyecto consiste en una aplicación Android para un sistema de domótica asistencial que controle los electrodomésticos mediante el reconocimiento de etiquetas a través de la cámara del dispositivo móvil. Los usuarios podrán mandar las órdenes para el funcionamiento de los aparatos a través de la voz. El actual documento tiene la finalidad de presentar el proceso completo llevado a cabo para la realización del proyecto.

Palabras clave: Android, domótica, QR, Smartphone, reconocimiento de voz, zXing.

Universidad
Carlos III de Madrid

Universidad
Carlos III de Madrid

Abstract

The project carried out describes the development of an Android application for an assistive automation system. It'll control appliances by recognizing labels through the camera of the mobile device. Users will be able of sending commands to operate the devices via voice. This can send commands to operate the devices via voice. The current document details the complete process needed to create the project.

Keywords: Android, automation, QR, Smartphone, voice recognize, zXing.

Universidad Carlos III de Madrid

Índice de contenidos

Índice de figuras

Índice de tablas

Capítulo 1. Introducción y objetivos

Este primer capítulo pretende aportar una visión general del desarrollo del proyecto, detallando la motivación y objetivos que han dado lugar a la realización del mismo. Se incluye un último punto para mencionar brevemente la estructura del documento a fin de facilitar su lectura.

1.1 Introducción

La realización de este proyecto se basa en dos pilares fundamentales: la domótica y los Smartphone.

A lo largo de los años los avances en los sectores de la informática, la industria y las telecomunicaciones han potenciado la evolución de la domótica hasta el punto en el que nos encontramos. Actualmente podemos hablar de una diversidad de posibilidades, abarcando desde todo tipo de comodidades a la hora de controlar los aparatos eléctricos hasta ser un aspecto fundamental en el ahorro energético, pasando por aspectos relacionados con la seguridad y las comunicaciones. Es por ello que los entornos de utilidad son básicamente cualquiera que podamos imaginar: universidades, empresas, hospitales, hoteles, viviendas particulares, etc. y por tanto los usuarios potenciales somos todos. Debido a la proliferación de este sector, la competitividad creada entre las empresas ha llevado a una oferta más económica que en sus inicios, de mejor calidad, con una mayor facilidad de uso y cada vez más adaptada a las demandas que van surgiendo. Así nos encontramos con un panorama en el que la domótica permite mejorar la calidad de vida adaptando el entorno a nuestras necesidades.

Por otro lado, tenemos lo que conocemos como Smartphone o teléfonos inteligente, definidos como dispositivos móviles cuyas características hardware y software le proveen de una mayor capacidad para almacenar datos y tener funcionalidades similares a las de un ordenador. La evolución de los Smartphone es un hecho evidente si pensamos en aquellos teléfonos de hace tan sólo 2 décadas que nada tienen que ver con los dispositivos que manejamos hoy, los cuales en ocasiones llegan a sustituir a los ordenadores personales. La facilidad de tener multitud de posibilidades en un dispositivo que cabe en el bolsillo ha dado lugar a que los usuarios de Smartphone cada día sean más numerosos llevando a su vez a un incremento en la demanda de aplicaciones que suplan las nuevas necesidades que van surgiendo.

En definitiva, este proyecto aprovecha las posibilidades que ofrecen ambas tecnologías para crear un sistema de domótica asistencial para Smartphone, en el que mediante la cámara del dispositivo se reconozca una etiqueta y el usuario pueda dar una sencilla orden por voz para activar una funcionalidad del aparato.

1.2 Motivación

Todos los días abrimos y cerramos un frigorífico, encendemos y apagamos una vitrocerámica, iniciamos el lavado en un lavavajillas…pero estas acciones que a priori parecen tan sencillas, suponen un gran problema para las personas con movilidad

reducida, que en muchos casos tendrán que recurrir a la ayuda de otras personas para realizar cualquiera de estas acciones cotidianas.

Esta barrera en accesibilidad a los electrodomésticos genera en las personas una falta de independencia total con todos sus problemáticas derivadas: frustración personal por la imposibilidad de desempeñar labores diarias, necesidad de otra persona que pueda ayudar, si no se dispone de ayuda familiar o conocidos, necesidad de una cantidad económica para pagar la asistencia de un profesional, etc.

Así, obtenemos la motivación para la realización de este proyecto: permitir a las personas con movilidad reducida llevar a cabo tareas por sí mismas sin necesidad de ayuda externa, dotándoles de una pequeña y a la vez gran independencia en su vida diaria.

1.3 Objetivos

El objetivo de este proyecto es desarrollar una aplicación que reconozca la etiqueta colocada en un electrodoméstico para posteriormente activar una opción propia del aparato mediante una orden por voz.

Consecuencia del objetivo principal aparecen los siguientes sub-objetivos más específicos:

- ✓ Reconocer los aparatos utilizando etiquetas (distancia usuario-etiqueta de 1.5mmín.)
- ✓ Almacenar en BBDD los datos referentes al electrodoméstico.
- ✓ Creación de una cadena de audio que reproduzca al usuario las opciones disponibles.
- ✓ Reconocimiento de voz para captar la opción que desea el usuario.
- ✓ Hacer una estructura de proyecto enfocada a que pueda ser la base de una línea futura más compleja.
- ✓ Realizar pruebas que verifiquen el funcionamiento de la aplicación.
- ✓ Adaptar la aplicación a posibles cambios descubiertos en la fase de test que puedan mejorar la experiencia de usuario.
- ✓ Hacer la aplicación funcional tanto de la manera tradicional interactuando con la pantalla como sin necesidad de tocarla, y siempre de un modo muy sencillo.
- ✓ Desarrollar el código de la aplicación de una manera clara, eficiente y estructurada.

1.4 Fases del desarrollo

Para desarrollar el proyecto se pueden definir 7 fases:

En primer lugar se realiza un estudio previo de las tecnologías posibles a utilizar, la plataforma y los resultados que dan otras aplicaciones con características similares. De esta manera podremos hacernos una idea de cuan viable es nuestro objetivo.

Como segundo paso se hace un análisis profundo de requisitos de usabilidad y sistema que debe cumplir la aplicación. Para ello se definen las limitaciones en movilidad de los usuarios como punto clave para el desarrollo del proyecto.

En función de las ideas establecidas en la fase anterior se procede a diseñar la arquitectura para dar una solución al conjunto de los requisitos. La finalidad es lograr cubrir todas las necesidades previamente definidas.

En cuarto lugar se procede a la implementación del código del proyecto.

Tras concluir la fase anterior se llevarán a cabo una serie de pruebas para verificar el funcionamiento y obtener resultados. Dichos resultados servirán para extraer posibles modificaciones que mejoren la usabilidad.

Por ello una vez terminada la fase de pruebas se procederá a realizar pequeños cambios que mejoren el funcionamiento del sistema o corrijan los errores detectados.

La séptima y última fase consiste en elaborar la memoria que tiene el lector en sus manos a fin de dejar reflejadas todas las etapas anteriores.

1.5 Estructura del documento

Este último apartado se expone un vistazo rápido al contenido de los diferentes capítulos del documento.

- **Capítulo 1**

El propósito del primer capítulo es aportar una visión global del proyecto, definiendo los objetivos y la motivación para su realización así como las etapas en las que se desarrollará.

- **Capítulo 2**

Expone la evolución de la plataforma y tecnologías en las que se basará la elaboración de este proyecto.

- **Capítulo 3**

En este capítulo en primer lugar, a partir de los objetivos se hace un análisis de los requisitos que deberá cumplir el sistema. A partir de ellos se toman las decisiones de diseño y se idea el flujo de navegación e interfaces de la aplicación que desembocará en la implementación final.

- **Capítulo 4**

Ahonda en la planificación realizada para la consecución de los objetivos y en base a las horas resultantes se elabora el presupuesto correspondiente.

- **Capítulo 5**

En este capítulo se comentan las conclusiones obtenidas de la elaboración del proyecto así como también tiene la intención de presentar una línea que complete y mejore el funcionamiento del sistema final.

- **ANEXOS, BIBLIOGRAFÍA YGLOSARIO**

Por último se presentarán los anexos con más información, entre la que encontraremos manuales de uso e información más detallada del estado del arte. Con la lectura de los contenidos en la presente memoria será suficiente para la comprensión del trabajo, no obstante se presentan los anexos por si el lector tuviera interés en conocer más. Finalmente se presentarán el glosario y la bibliografía.

Capítulo 2. Estado de la cuestión

Este capítulo habla de la evolución de las tecnologías empleadas, así como de la plataforma, tipos de etiquetas y librerías utilizadas para el desarrollo del la aplicación, para explicar al lector las tecnologías utilizadas que también serán comentadas en este apartado.

2.1 Dispositivos móviles

2.1.1 Definición y características

En primera instancia podríamos asociar el concepto de dispositivo móvil con el de teléfono móvil. Sin embargo el primero es un término utilizado para un abanico de terminales, como los que muestra la figura bajo estas líneas entre otros como los dispositivos reproductores de audio, PDAs o cámaras fotográficas.

Figura 1. Dispositivos móviles

Sin embargo ciñéndonos a su definición podemos enumerar las siguientes características básicas:

- un aparato de pequeño tamaño,
- posee algunas capacidades de procesamiento,
- tiene conexión permanente o intermitente a una red,
- dotado de una memoria limitada,
- se diseña para una funcionalidad específica; pero tener otras más generales,

Además de estas características básicas, podemos también considerar algunas otras como no ser necesariamente extensibles y actualizables, en pocos años el usuario deberá cambiarlo, barato y de fácil manejo, por lo que no requieren usuarios expertos, etc. Encontramos más información en el Anexo I.

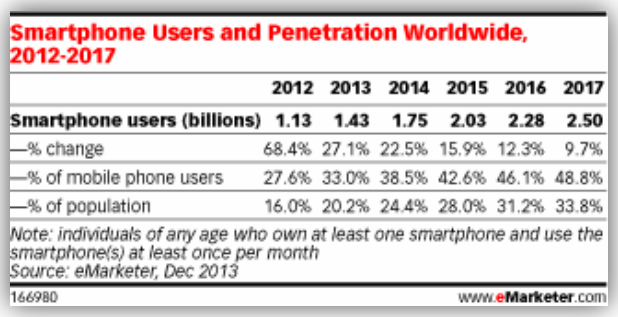
2.1.2 Smartphone

Dentro de los dispositivos móviles, los teléfonos inteligentes o Smartphone son teléfonos móviles construidos sobre una plataforma informática móvil, con una mayor capacidad de almacenar datos y realizar actividades similares a una minicomputadora y con una mayor conectividad que los teléfonos móviles convencionales. El término “inteligente” es utilizado comercialmente para referirse a la posibilidad de utilizar el teléfono como un ordenador de bolsillo, llegando incluso a sustituir el ordenador personal en algunos casos. Por tanto esta idea de teléfono difiere en gran medida del concepto tradicional. (1)

Entre otras mejoras se pueden mencionar la Geolocalización GPS, el acelerómetro, el teclado QWERTY, navegadores web, clientes de correo, aplicaciones ofimáticas,

reproductores de contenido audiovisual, la posibilidad de descargar e instalar aplicaciones adicionales, la pantalla táctil, cámara y reproductor de vídeos/mp3, el programa de agenda, administrar contactos, etc. De hecho para ser reconocido popularmente como Smartphone suelen considerarse dos características fundamentales: la pantalla táctil y el correo electrónico, ya que desde 2007 todos los modelos existentes y anunciados poseen ambas cualidades. Pero quizá la mejora más importante para los Smartphone viene de la mano de la conexión a internet, primeramente mediante Wi-Fi o 3G y desde hace ya unos años el 4G aumenta la velocidad de conexión (hasta 100 Mbps teóricos de bajada), permitiendo videollamadas HD sin cortes, un streaming bastante más fluido y acceso a los contenidos en la nube de manera más rápida entre otros; aunque aún está teniendo problemas de cobertura y no todos los terminales lo soportan.

Dado el conglomerado de características ofrecidas, el crecimiento del número de usuarios de Smartphone es destacable. Según un estudio realizado por eMarketer, los usuarios de Smartphone superaron la cifra de 1.000 millones y se prevé que a al cierre del 2014 se habrá alcanzado 1.750 millones de usuarios. eMarketer señala también que esta tendencia ascendente nos llevará a tener 2.500 millones de usuarios de Smartphone en 2017. Podemos observar estos datos en la siguiente figura:



Smartphone Users and Penetration Worldwide, 2012-2017

	2012	2013	2014	2015	2016	2017
Smartphone users (billions)	**1.13**	**1.43**	**1.75**	**2.03**	**2.28**	**2.50**
—% change	68.4%	27.1%	22.5%	15.9%	12.3%	9.7%
—% of mobile phone users	27.6%	33.0%	38.5%	42.6%	46.1%	48.8%
—% of population	16.0%	20.2%	24.4%	28.0%	31.2%	33.8%

Note: individuals of any age who own at least one smartphone and use the smartphone(s) at least once per month
Source: eMarketer, Dec 2013

166980 www.eMarketer.com

Figura 2. Usuarios de Smartphone

No obstante es imposible obviar las restricciones que tienen estos dispositivos a pesar de las importantes mejoras mencionadas. El motivo es que el reducido tamaño de estos dispositivos conlleva ineludiblemente a limitaciones de hardware que marcan de una forma clara sus diferencias con los ordenadores convencionales. Así por ejemplo es directo ver que la ventajosa portabilidad de la que les provee su tamaño se transforma en impedimento a la hora de hablar del tamaño de las pantallas. Otros aspectos son la menor capacidad del procesador, las restricciones en la memoria RAM y en el almacenamiento de datos persistentes y la necesidad de adaptar el consumo de energía a una batería de pequeñas dimensiones.

Ventajas y desventajas son de obligado conocimiento para desarrollar un software de calidad, que aproveche al máximo las características de los Smartphone teniendo siempre muy en cuenta las capacidades reales del dispositivo.

2.1.3 Sistemas operativos

Habida cuenta de las innumerables posibilidades que ofrecen los dispositivos móviles, son variados los sistemas operativos que se han ido instaurando a lo largo de los últimos años. (2)Los principales podemos encontrarlos en el Anexo I, y aquí citaremos sólo el elegido: Android. No obstante se recomienda la lectura del anexo para conocer las características y diferencias entre todos ellos.

- **Android**

Android es el S.O. de Google para Smartphone, PDA y terminales móviles. Su línea de desarrollo parte de la idea de un entorno abierto para cualquier programador y fabricante y ser así un sistema libre. Su versatilidad queda patente en la gran aceptación que ha tenido por parte los fabricantes (como Samsung, LG, HTC, Motorola…), lo cual se traduce en una gran acogida por parte de los usuarios, quienes disponen de una gran variedad de tamaños de pantallas, cámaras con diferentes resoluciones, terminales más modestos, y un sinfín de opciones.

Figura 3. Android

Por otro lado, la experiencia de usuario es bastante buena y la oferta de aplicaciones en el Google Play es amplia y en su mayoría gratuita (frente a iOS, su principal competidor), lo que acerca aún más este S.O. a la inmensa mayoría de usuarios.

- Ventajas
  - Google Maps: un GPS gratuito.
  - El costo de los Smartphone con este S.O.
  - La gama de aplicaciones es muy extensa.
- Inconvenientes
  - El mayor consumo de la batería.
  - La experiencia de usuario puede variar entre diferentes terminales.

Por todo ello, hoy en día Android sigue siendo el principal S.O. en Smartphone frente a todos sus competidores. Podemos ver en la figura bajo estas líneas la evolución en las ventas de Smartphone según sus S.O. y la previsión para los próximos años. Si comparamos el número de usuarios que eligen Smartphone con Android frente al resto de sistemas, los porcentajes son contundentes:

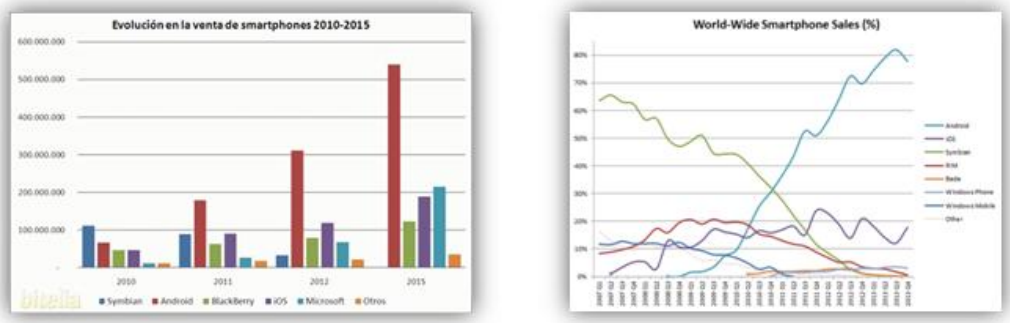


Figura 4. Evolución ventas Smartphone 2010-2015 y evolución de los S.O. en Smartphone

2.2 Plataforma Android

2.2.1 Arquitectura

La arquitectura Android se basa en un kernel Linux 2.6, similar al que puede incluir cualquier distribución de Linux, sólo que adaptado a las características del hardware en el que se ejecutará Android, es decir, para dispositivos móviles. Este tipo de núcleo aporta rapidez, fiabilidad y seguridad. El kernel se encarga de gestionar los diferentes recursos del teléfono (energía, memoria, etc.) y del S.O. en sí: procesos, elementos de comunicación (*networking*), etc. La mayor parte de la plataforma Android está disponible bajo licencia de software libre de Apache y otras licencias de código abierto. Bajo estas líneas vemos una figura que ilustra la estructura de un sistema Android. (3)

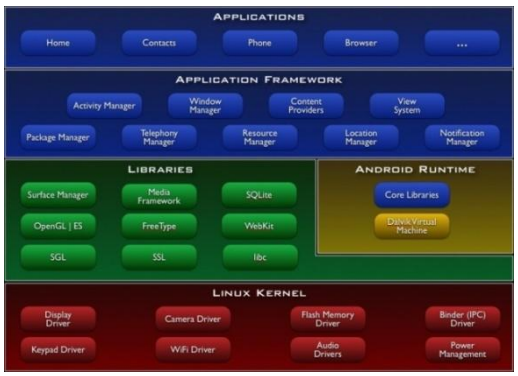


Figura 5. Arquitectura plataforma Android

Fundamentalmente podemos ver lo descrito anteriormente, que la base es un núcleo Linux, sobre el que están las librerías (OpenGL como motor gráfico y SQLite para BBDD, entre otras) y la máquina virtual Dalvik que hace que con una sola compilación las aplicaciones estén listas para distribuirse garantizando que cualquier dispositivo Android pueda ejecutarlas. Después tenemos el framework de aplicaciones que está formado por todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones, y por último la capa de aplicaciones que engloba tanto las que tienen interfaz como las que no, las propias del sistema y las descargadas y por ejemplo la aplicación de 'Home' que lista las aplicaciones y permite lanzarlas, estaría contenida en esta capa. Toda esta información y las otras características interesantes se encuentran mucho más detalladas en la parte correspondiente del Anexo I.

- ✓ Android dota a fabricantes, desarrolladores y usuarios de una libertad total para controlar como desee su terminal. Es por ello que se irgue como la opción seleccionada por una inmensa mayoría, por lo que a la hora de decidir qué plataforma elegir para nuestra aplicación, si tenemos intención de que sea accesible para un elevado número de usuarios con todo tipo de teléfonos y marcas, Android será la decisión más factible.

2.2.2 Ciclo de vida

El ciclo de vida de una aplicación Android es bastante diferente al ciclo de vida de una aplicación en otros S.O., como Windows. La mayor diferencia es que, en Android el ciclo de vida es controlado principalmente por el sistema, en lugar de ser controlado directamente por el usuario.

Una aplicación en Android va a estar formada por un conjunto de elementos básicos de interacción con el usuario, conocidos como actividades. Además de varias actividades una aplicación también puede contener servicios. El sistema va a mantener una pila con las actividades previamente visualizadas, de forma que el usuario va a poder regresar a la actividad anterior pulsando el botón atrás. (4)

Android es sensible al ciclo de vida de una actividad, por lo tanto, veamos los eventos posibles:

Activa (*Running*): La actividad está encima de la pila, está visible y tiene foco.

Visible (*Paused*): La actividad es visible pero no tiene el foco. Se alcanza este estado cuando pasa a activa otra actividad con alguna parte transparente o que no ocupa toda la pantalla. Cuando una actividad está tapada por completo, pasa a estar parada.

Parada (*Stopped*): Cuando la actividad no es visible. El programador debe guardar el estado de la interfaz de usuario, preferencias, etc.

Destruida (*Destroyed*): Cuando la actividad termina al invocarse el método *finish()*, o es matada por el sistema.

Cada vez que una actividad cambia de estado se van a generar eventos que podrán ser capturados por ciertos métodos de la actividad. A continuación se muestra un esquema que ilustra los métodos que capturan estos eventos.

onCreate(Bundle): Se llama en la creación de la actividad. Se utiliza para realizar todo tipo de inicializaciones, como la creación de la interfaz de usuario o la inicialización de estructuras de datos. Puede recibir información de estado de la actividad (en una instancia de la clase Bundle.

onStart(): Nos indica que la actividad está a punto de ser mostrada al usuario.

onResume(): Se llama cuando la actividad va a comenzar a interactuar con el usuario. Es un buen lugar para lanzar las animaciones y la música.

onPause(): Indica que la actividad está a punto de ser lanzada a segundo plano, normalmente porque otra actividad es lanzada. Es el lugar adecuado para detener animaciones, música o almacenar los datos que estaban en edición.

onStop(): La actividad ya no va a ser visible para el usuario y si hay poca memoria, la actividad puede destruirse sin llamar a este método.

onRestart(): Indica que la actividad va a ser representada después de pasar por *onStop()*.

onDestroy(): Se llama antes de que la actividad sea totalmente destruida. Por ejemplo, cuando el usuario pulsa el botón "volver" o cuando se llama al método *finish()* y si hay poca memoria, es posible que la actividad se destruya sin llamar a este método.

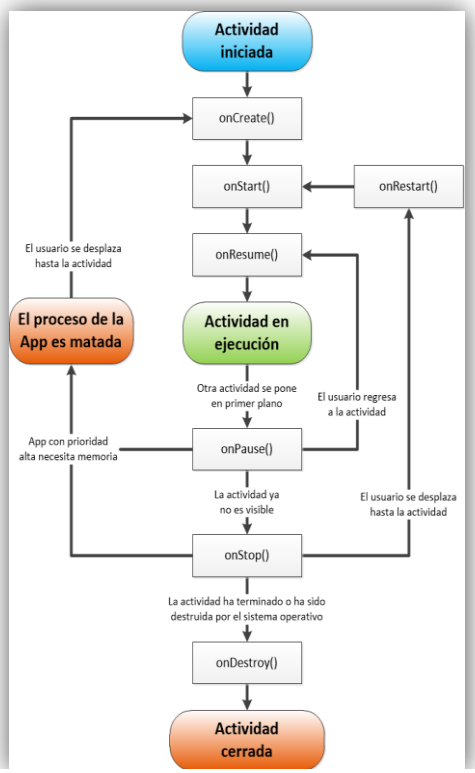


Figura 6. Ciclo de vida de una aplicación Android

2.2.3 Aplicaciones Android y discapacidad

Desde llegada de Android a los Smartphone, el número de aplicaciones en el ha crecido de manera significativa. A día de hoy prácticamente cualquier cosa que queramos hacer podemos encontrar una aplicación para ello, lo cual es extensible al concepto de aplicaciones de accesibilidad. Es tal la utilidad de los Smartphone Android para las personas con discapacidad que existe un portal web llamado 'Amóvil' que ayuda a los usuarios a identificar dispositivos, productos de apoyo, aplicaciones compatibles y servicios adaptados a sus necesidades y preferencias. La Fundación ONCE y la Fundación Vodafone España han renovado el convenio de colaboración para el fortalecimiento de dicho portal. También existe otro portal llamado Androidaccess.net que es un repositorio de aplicaciones enfocadas a la accesibilidad.

Veamos a modo representativo una selección de 4 aplicaciones de este tipo (5):

- **SignARTE**

Es una aplicación para Android creada por la Fundación CNSE para la Supresión de Barreras de Comunicación, que permite a las personas sordas localizar los espacios culturales accesibles en cualquier ciudad española para que la visiten a través del teléfono móvil o tablet. Recoge información en lengua de signos sobre cines, teatros, bibliotecas, museos, etc. que ofrecen servicios de accesibilidad como intérpretes de lengua de signos, servicios de

Figura 7. SignArte

emergencia visuales, etc. Incorpora la función “Cerca de mí” que muestra a los usuarios la oferta cultural accesible en un radio de hasta 100km a partir de su ubicación. Esta útil guía que rompe las barreras de comunicación está disponible de forma gratuita.

- **BrailleBack**

Figura 8. BrailleBack

Es una aplicación para ciegos o personas con dificultad de visión, que fusiona el soporte de voz TalkBack con el braille, permitiendo conectar un dispositivo de braille a nuestro teléfono por bluetooth y transferir todo lo que pasa en la pantalla. Para configurarla sólo hay que verificar la conexión Bluetooth del dispositivo de Braille y emparejarlo con el móvil. Después ir a “Ajustes” y dentro de “Accesibilidad” activar el checkbox de BrailleBack. Así, de un modo sencillo la aplicación está lista para ser utilizada.

- **BigLauncher**

Figura 9. BigLauncher

Esta aplicación ha sido diseñada pensando en los mayores y en aquellos usuarios con problemas de visión o problemas motrices. Está disponible para Android 2.1 o superior por 10$; pero existe una versión demo gratuita. BigLauncher reemplaza la interfaz de usuario para ofrecer la máxima legibilidad y facilidad de uso. Facilita atajos grandes para todas las aplicaciones, cambia tonalidades y tamaños de letra para su mejor visionado, tiene lector de pantalla, etc. Tiene la opción de controles alternativos para manejarla desde un teclado de hardware o con la interfaz para silla de ruedas Tecla. Esto permite a los usuarios con parálisis tener un control completo y preciso de su Smartphone sin tocar la pantalla.

- **Accessibility**

Figura 10. Accessibility

Es una aplicación para personas con movilidad reducida que les permite visualizar en la pantalla del móvil los espacios y lugares adaptados que tiene a su alrededor con tan sólo enfocar la cámara. A través de la realidad aumentada y la Geolocalización en tiempo real la aplicación aporta información sobre aparcamientos reservados, gasolineras, ambulatorios, cajeros automáticos, etc. La información se clasifica en dos tipos: la que ha sido verificada por la Federación de Asociaciones de Personas con Discapacidad Física y Orgánica de la Comunidad de Madrid y la que no, que es la referida a los datos que aporta la EMT y la aportada por los usuarios. La aplicación funciona tanto en

exteriores como en interiores aunque en este último caso la precisión disminuye al dificultarse la recepción de datos por GPS. También es posible para el usuario marcar con estrella los lugares favoritos para después poder acceder a los sitios más utilizados por el usuario. Como característica a destacar incluye una detección de fuerte impacto del terminal o detección de deceleración brusca, tras lo cual ofrece la posibilidad de conectar con el 112 por si fuera necesario avisar a los servicios de emergencia.

2.3 Domótica

2.3.1 Definición y características

El término domótica viene de dos palabras: *domus* que significa casa en latín y tica de automática, que significa que funciona por sí sola. Por tanto, entendemos por domótica el conjunto de sistemas capaz de automatizar una vivienda, aportando servicios de gestión energética, seguridad, bienestar y comunicación, y que pueden estar integrados por medio de redes interiores y exteriores de comunicación, cableadas o inalámbricas y cuyo control goza de cierta ubicuidad, desde dentro y fuera del hogar. Se podría definir como la integración de la tecnología en el diseño inteligente de un reciento cerrado. (6)

El sistema funcionaría así: se recoge la información proveniente de unos sensores (entradas), se procesa y se emiten las órdenes a unos actuadores (salidas). La integración de la red de control del sistema domótico está regulada por la instrucción ITC-BT-51 Instalaciones de sistema de automatización, gestión técnica de la energía y seguridad para viviendas y edificios. Los mandos del sistema domótico funcionan por programas de barrido y cada botón tiene una imagen y un texto descriptivo que define para qué sirve el botón. Los dispositivos se agrupan en categorías para diferenciar los aparatos que regula.

- En el Grupo 1 los programas relativos a ocio (que controlan la televisión, los equipos de música, la prensa y páginas web, juegos, libros, etc.).

- En el Grupo 2 están todos los controles del PC, teléfono y electrodomésticos.

- En el Grupo 3 están los programas de servicios domóticos de uso diario y cotidiano. Aquí entran el control de temperatura, luces, ventanas, puertas, cama, etc.

Por lo tanto son muchas y variadas las aplicaciones que tiene la domótica, como por ejemplo el ahorro energético que puede suponer, el confort que proporciona, así como una gran accesibilidad (7) que puede aportar a personas con impedimentos en la movilidad, como por ejemplo interfaces inalámbricos que permiten controlar aparatos solo con un movimiento de cabeza (FATRONIK). Una explicación más detallada de estas funcionalidades la encontramos en el Anexo I.

- ✓ Tras las características descritas, es directo pensar en las posibilidades que nos ofrece la domótica para hacer nuestro entorno un lugar apto para todos con independencia de las discapacidades en movilidad. Por ello, si unimos Android con la domótica podemos dotar a los usuarios de un control si no total, al menos lo más independiente posible.

2.4 QR

2.4.1 ¿Qué es un código QR?

El acrónimo QR, "Quick Response", significa código de respuesta rápida. Es un código muy útil para almacenar información en una matriz de puntos o un código de barras bidimensional. Fue ideado en 1994 por la compañía japonesa Denso Wave, subsidiaria de Toyota. El estándar japonés para códigos QR (JIS X 0510) fue publicado en enero de 1998 y su correspondiente estándar internacional ISO(ISO/IEC18004) fue aprobado en junio de 2000.

Figura 11. Ejemplo de código QR

Los creadores (un equipo de dos personas en Denso Wave, dirigido por Masahiro Hara)tenían como objetivo que el código permitiera que su contenido se leyera a alta velocidad. Los códigos QR son muy comunes en Japón y de hecho son el código bidimensional más popular en ese país. (8)

2.4.2 Características y reconocimiento

Un QR se caracteriza fundamentalmente por los 3 cuadrados de las esquinas, los cuales permiten detectar la posición del código al lector, un conjunto que contiene los datos y la corrección de errores y un subconjunto de cuadrados pequeños que aportan información sobre la versión del código. (9)

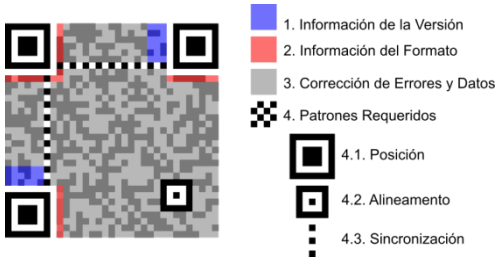


Figura 12. Estructura código QR

Aunque inicialmente se usó para registrar repuestos en el área de la fabricación de vehículos, hoy los códigos QR se usan para innumerables aplicaciones. La inclusión de software que lee códigos QR en teléfonos móviles, ha permitido nuevos usos orientados al consumidor, que se manifiestan en todo tipo de funcionalidades como el dejar de tener que introducir datos de forma manual en los teléfonos.

Un detalle importante sobre el código QR es que, a diferencia de otros formatos de códigos de barras bidimensionales como el BIDI, su código es abierto y sus derechos de patente (propiedad de Denso Wave) no son ejercidos.

Veamos a continuación características que influirán decisivamente en el reconocimiento de las QR:

- **Versión del código QR**

A la hora de generar QR tenemos que decidir qué versión y qué nivel de corrección de errores emplearemos. En cuanto a las versiones nos referimos a QR v1 al que consta de 21 módulos o columnas por cada lado. A medida que aumenta la versión, el número de módulos aumenta de 4 en 4, así un QR v2 se compone de una matriz 25x25. En los extremos de las versiones tenemos:

- el v40 como máximo, formado por 177 módulos por lado,
- y existe un microcódigo QR, que es una versión más pequeña del estándar del código QR y está diseñado para aplicaciones que tengan una habilidad menor en el manejo de escaneos grandes. Hay diferentes versiones de micro código QR. La más grande de ellas puede contener hasta 35 caracteres y si nos fijamos en la imagen, vemos que únicamente tiene un patrón de posición, por lo que tendrá una utilidad específica y limitada; pero que en situaciones con poca información puede resultar provechoso.

Figura 13. Microcódigo QR

- **Nivel de corrección de errores**

En cuanto a la capacidad de corrección de errores existen varios niveles que vemos en la siguiente tabla:

Capacidad de corrección de errores	
Nivel L	7% de las claves se pueden restaurar
Nivel M	15% de las claves se pueden restaurar
Nivel Q	25% de las claves se pueden restaurar
Nivel H	50% de las claves se pueden restaurar

Tabla 1. Capacidad de corrección de errores

Si miramos la imagen bajo estas líneas podemos apreciar un código con nivel de corrección L, que tras borrar algunos datos y agregar otros se puede reconocer sin errores es detectado correctamente. De ello podemos deducir que ya con el nivel mínimo los resultados son más que aceptables.

Figura 14. Corrección de errores en códigos QR

→El número de caracteres que puede almacenar un QR varía en función de la versión, de si son caracteres alfanuméricos o no y del nivel de corrección. De esta manera por ejemplo para la v1 con nivel de corrección L, hablamos de 17 caracteres alfabéticos o 41 caracteres numéricos.

- **Personalización de QRs**

Figura 15. QRs personalizados

Aunque los QR pueden ser dinámicos (cambiar la imagen de fondo cada día), contener fondos con imágenes customizadas, variar los colores, en definitiva personalizarlos todo lo que queramos, no es recomendable hacer uso de estas características si nuestro interés es meramente el reconocimiento (y no el marketing), ya que dificultan la lectura en gran medida. Así lo óptimo será utilizar los códigos en blanco y negro para que no haya impedimentos a la hora de activar el reconocimiento.

- **Iluminación del entorno**

Para el reconocimiento de un QR es fundamental la iluminación del entorno en el que se encuentre, cuanto mayor iluminada esté la sala mejor será el reconocimiento. La influencia práctica de esta característica la veremos al analizar el reconocimiento de QRs.

- **Tamaño de código QR**

De igual manera que la iluminación, el tamaño de la QR será determinante para su reconocimiento. La fórmula básica que relaciona tamaño y distancia de reconocimiento es la siguiente,

Tamaño mínimo QR = Distancia de reconocimiento / 10 (1)

Aunque como veremos a continuación, el resto de factores también influirán en la variación de esta distancia, positiva o negativamente; por lo que es preciso hacer un análisis más profundo.

❖ RECONOCIMIENTO DE QRs

En último lugar tratamos el aspecto quizá más importante, la distancia de reconocimiento. Para optimizarla es importante, que nuestro QR no tenga colores de fondo y el contraste entre los módulos sea el mayor posible, es por ello que elegiremos el blanco y el negro para su representación. La iluminación de la sala en la que se instale la etiqueta, puede hacer variar en varios centímetros la distancia de reconocimiento. Tamaño de la QR y versión son fundamentales a la hora de determinar la distancia a la que se podrá escanear. A iguales dimensiones un código v1 tiene menor densidad de datos que uno de versión 10, por lo que será escaneado con mayor facilidad.

Como veíamos antes, la distancia de reconocimiento se relaciona directamente con el tamaño de la QR siguiendo la relación (1). Así por ejemplo para una distancia de 50 cm, tendremos que disponer de una QR de 5x5 cm.

No obstante, esta fórmula está demasiado simplificada para contemplar todos los aspectos mencionados antes que influyen en la distancia de reconocimiento, y por tanto no refleja la realidad. De esta manera, llegamos a la siguiente fórmula, más específica:

Tamaño mínimo QR Code = (DE / FD) * FD (2)

donde,

- DE=Distancia de escaneo
- FD=Factor distancia: Inicio de un factor de 10, entonces reducir en 1 por cada uno de poca luz en el ambiente de exploración
- FD=Factor de densidad de datos: Se divide el número de módulos del QR emplear entre 25 para normalizar con QR v2 (que es una matriz de 25x25)

→Pongamos un ejemplo para verlo mejor:
Queremos almacenar información en un QR v2 (25x25), la iluminación será un poco oscura y el tamaño del código deberá ser de 27x27cm. Apliquemos la fórmula anterior:

27=[distancia/(10-1)] * 25/25;

Despejando la distancia obtenemos un valor de 2.5 m.

Si no hubiéramos tenido en cuenta todos los factores, siguiendo la fórmula 1 hubiéramos llegado a un resultado de 27=distancia/10; es decir a una distancia de 2.7m, resultado que en la realidad no se hubiera cumplido y que además ya a priori es mayor que el obtenido con la expresión completa.

2.4.3 Generación y aplicaciones

Generar códigos QR es gratuito y sencillo ya que hay multitud de páginas web desde las que es posible crearlos y descargarlos rápidamente. Algunas de ellas son: www.goqr.me, www.beqrious.com/qr-code-generator/ owww.qrcode.kaywa.com. Además ya existen un gran número de aplicaciones lectoras para Smartphone como Beetag Reader, UpCode Reader o Barcode Scanner entre otras muchas, y su funcionamiento es tan sencillo como: apuntar la cámara, reconocer y mostrar la información que contiene.

Figura 16. Flujo de reconocimiento de un código QR

Todo ello hace que las aplicaciones de las QR sean innumerables y que cada día sigan aumentando. Podemos enumerar las siguientes a modo de ejemplo:

- Publicidad y Márketing, por ejemplo "Prenatal" lo utiliza en sus escaparates para dar la bienvenida al cliente ofreciéndole un desfile de moda en vídeo online.
- Guardar contactos en la agenda del Smartphone sin tocar la pantalla.
- El periódico "¿QUÉ?" fue pionero utilizando las QR con información de sus noticias.
- Sistemas de ticket o entradas móvil.
- Servicios de logística.

2.4.4 Diferencias con otros códigos

- **Códigos de colores**

Los códigos de colores suponen varios problemas frente a los QR:

- No almacenan información, por lo que tras su reconocimiento habría que ir a una BBDD que contuviera toda la información.
- Las distintas tonalidades, resoluciones de las cámaras de los Smartphone no los hacen universales, por lo que habría problemas de unicidad en los códigos además de que no podríamos detectar errores.

- **NFC**

Los códigos NFC están basado en la emisión y recepción de datos en alta frecuencia a corta distancia, en torno a los 10 cm. de alcance. Sin embargo, como ya vimos en las

características de un QR, las distancias de reconocimiento mejoran ampliamente este valor.

- **Marcadores de realidad aumentada**

Con los marcadores de realidad aumentada estamos condicionados por la aplicación que los use, ya que se diseñan únicamente para cada aplicación. Por el contrario los QR son universales y cualquiera puede generarlos para ser reconocidos por cualquier aplicación.

- **Códigos de barras**

Este último tipo sólo puede contener una limitada secuencia de dígitos (no contenido alfanumérico) además de no poseer corrección de errores, por lo que en superficies rugosas se dificulta en gran medida su lectura. Aunque hay multitud de tipos de códigos de barras, por regla general, el lector debe estar bastante cerca del código.

Todas las anteriores características restrictivas son determinantes a la hora de compararlos con los códigos QR.

2.5 Reconocimiento de imágenes

2.5.1 Introducción

El reconocimiento de imágenes sigue el siguiente proceso:

Los sensores captan los datos, se extraen las características que los definen y finalmente se clasifican siguiendo el algoritmo diseñado. Básicamente el reconocimiento se hace siguiendo el siguiente esquema:

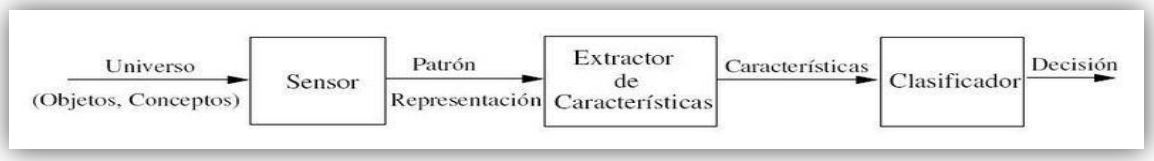


Figura 17. Reconocimiento de imágenes

Aunque todas las etapas son importantes, el punto esencial es la clasificación, ya en función de ella se tomará una decisión. Por ejemplo, se pueden clasificar imágenes en tomates y pelotas de tenis, considerándose tomates aquellas que superen un porcentaje de píxeles rojos y pelotas de tenis si los píxeles son amarillos. De igual modo que se puede clasificar en función del color, es posible elegir cualquier tipo de característica: la forma, el fondo, formas dentro de otras formas (un círculo dentro de un cuadrado para reconocer lavadoras por ejemplo), con patrones predefinidos en la imagen, etc.

Es imperativo pensar en la cantidad de posibilidades que nos ofrece el mundo del reconocimiento de imágenes, entre ellas podemos contemplar:

- Reconocimiento de caracteres,
- Reconocimiento facial y de huellas
- Detección de enfermedades
- Interpretación de imágenes aéreas o vía satélite

2.5.2 Librerías

Existen multitud de librerías para el reconocimiento de imágenes debido a su variada utilidad. En este capítulo mostraremos sólo las que han sido contempladas para el desarrollo de este proyecto.

- **OpenCV**

Figura 18. OpenCV

En 1999, Intel sacó la primera versión alfa de esta librería libre de visión artificial. Desde el inicio ha tenido un abanico de aplicaciones que van desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos. Esto ha sido posible gracias a que se publicó bajo la licencia BSD, que permite utilizar la librería libremente con fines comerciales y de investigación bajo las condiciones en ella expresadas. Existen versiones de la librería para GNU/Linux, Mac OS X y Windows y contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como el reconocimiento de objetos o la visión robótica. Pretende ser fácil de utilizar y aprovechar al máximo las características del código en C y C++ y de los procesadores multi-núcleo para conseguir una alta eficiencia.

Aplicaciones de esta librería son: los sistemas de vigilancia de vídeo además de haber sido también utilizada en el sistema de visión del vehículo no tripulado Stanley de la Universidad de Stanford (ganador en el año 2005 del Gran desafío DARPA)

- **Zbar**

Es una librería de software de código abierto para la lectura de códigos de barras a partir de diversas fuentes, tales como transmisiones de video, archivos de imagen y sensores de intensidad primas. Es compatible con varios tipos de etiquetas como EAN-13/UPC-A, Code 38, Interleaved 2 de 5 y códigos QR.

Su flexibilidad y funcionamiento autónomo hace que el escaneo y la decodificación sean sencillas a la hora de integrarlas con un proyecto. Sin embargo a pesar de que algunas fuentes la señalan como más rápida y fácil de utilizar que otras como zXing, la versión para Android no da demasiados buenos resultados en cuanto a distancias de reconocimiento.

Zbar es multiplataforma, está disponible para Linux/Unix, Windows, iPhone, gracias al escaneo de los flujos de video en tiempo real consigue altas velocidades de

procesamiento, no está limitado a un tipo de imágenes y es adecuada para aplicaciones que utilizan procesadores de bajo coste. Por ello tiene aplicación en los móviles, la venta al por menor, el seguimiento de inventarios, etc. (10)

- **ZXing**

Figura 19. ZXing

La librería zXing es un proyecto open-source que ofrece soporte para la lectura y decodificación para códigos 1D y 2D, entre los que se encuentran la gran mayoría de códigos de barras, códigos BIDI o QR en múltiples plataformas, como Java, C++, Android, iOS incluso J2ME.

A continuación vemos una figura con los tipos de formatos que reconoce zXing:

1D product	1D industrial	2D
UPC-A	Code 39	QR Code
UPC-E	Code 93	Data Matrix
EAN-8	Code 128	Aztec (beta)
EAN-13	Codabar	PDF 417 (beta)
	ITF	
	RSS-14	
	RSS-Expanded	

Figura 20. Tipos de formatos reconocidos por zXing

Su funcionamiento consiste en detectar los marcadores dentro del QR y muestra en la pantalla multitud de cuadrados verdes que tratan de buscar sus homólogos en el QR para decodificarlos. (11)

✓ zXing es la librería utilizada en este proyecto para el reconocimiento de etiquetas debido a sus características desarrolladas aquí, además de su óptimo funcionamiento.

2.6 Reconocimiento de voz

2.6.1 Introducción e implementación

En este proyecto se tendrán en cuenta dos aspectos del sonido en las aplicaciones Android: la síntesis de la voz y el reconocimiento de la misma.

- Síntesis del sonido

La síntesis de voz consiste en la creación de ondas de sonido artificiales semejantes al habla humana, lo que en inglés se conoce como Text To Speech, TTS, conversión de texto a voz. Los sistemas de síntesis de voz tienen dos partes diferenciadas:

- El front-end que se encarga de recoger el texto y procesarlo, lo que se conoce como normalización del texto o pre-procesado, se traducen los números a palabras, las abreviaturas y se interpretan los signos de acentuación. En esta parte se separan las palabras y se les asigna una transcripción fonética.
- El back-end que consiste en crear las ondas de sonido mediante las configuraciones que han tenido lugar en la fase de pre-procesado.

Android desde su versión 1.6 tiene TTS nativo capaz de sintetizar textos en varios idiomas de una manera rápida y eficaz. Algunas aplicaciones que utilizan TTS son: *Easy Text To Speech*, que "lee en voz alta" cualquier texto que se le pase y es gratuita, o *IVONA Text-to-Speech HQ* cuya instalación gratuita en el Smartphone permite al usuario escuchar los textos u opciones de sus aplicaciones pudiendo elegir entre 13 idiomas diferentes.

- Reconocimiento de voz

El reconocimiento del habla o voz, es el sistema inverso del TTS. Lo que hace es transcribir un mensaje oral en texto basándose en la comparación con un modelo acústico recogido en una base de datos. Su implantación comercial la apreciamos en las compañías telefónicas, el acceso a discapacitados visuales, etc.

De la misma manera que el TTS, también es posible utilizar el reconocimiento de voz en Android, el cual llama a Google VoiceSearch disponible en los terminales Android (y si no la descarga). De hecho en las versiones de Android posteriores a 4.0 si abrimos Google Voice Search y decimos "OK Google" (12), tendremos a nuestra disposición todo tipo de alternativas a realizar en nuestro Smartphone a través del reconocimiento de voz: podremos abrir una aplicación diciendo "Abrir <nombreAplicación>", preguntar por el tiempo, añadir un alarma, llamar a un contacto, etc. Por ahora, esta opción está disponible en francés, alemán e inglés.

Son variadas las aplicaciones Android que utilizan el reconocimiento de voz, entre las que se encuentran: *Traductor*, que básicamente es un diccionario gratuito en varios idiomas y traduce lo que recoge del habla del usuario al idioma que se seleccione, *Lyngo*, que permite escribir SMS a partir de un dictado de voz, realizar búsquedas en Internet, o incluso actualizar su estado de Facebook, aunque esta última cuesta 1.99€.

2.7 Tecnologías utilizadas

Aunque a lo largo de este capítulo se ha ido desgranando las tecnologías trabajadas, en este apartado se resumen al lector para que tenga una visión general de las mismas.

❖ Smartphone

Se opta por implementar el proyecto para Smartphone frente a otros dispositivos como una tablet fundamentalmente por su reducido tamaño. El motivo es las

características del usuario, ya que al ir en una silla de ruedas y no tener movilidad para utilizar la aplicación, el móvil irá instalado en un soporte diseñado para la silla y no sería lógico utilizar una tablet puesto que su colocación sería más aparatosa. Además también se descartan otros dispositivos como una PDA por sus escasas funcionalidades comparadas con las de un Smartphone. El modelo del que se dispone para la emulación del código y las pruebas finales es el Samsung Galaxy S.

- Android

Dadas las características explicadas 2.2 y en Anexo I y la comparativa con otras plataformas como iOS, se opta por la plataforma Android por ser la que mayor número de usuarios utiliza, para así tratar de llegar al mayor número de personas posible. Otro argumento de peso es el hecho de que sea una plataforma libre, ya que en contraposición con iOS por ejemplo cuyos programadores deben pagar una cuota para tener licencia de desarrollador, Android es completamente gratuita y existen innumerables librerías también libres para implementar un sinfín de funcionalidades. Además, las últimas versiones de la plataforma tienen en nativo tanto el TextToSpeech (para convertir una cadena de texto en un audio capaz de ser reproducido) como el reconocimiento de voz a través de Google VoiceSearch, disponible hoy en todos los dispositivos Android; y ambas características son básicas para el desarrollo de este proyecto.

- Eclipse

Eclipse es un entorno de desarrollo que facilita mucho la tarea del programador. Haciendo referencia a un objeto y con una simple combinación de teclas (ctrl+spacio) el IDE nos muestra un listado de métodos disponibles y su cabecera, por lo que si no recordamos cómo utilizar algo, de esta manera lo tendremos accesible al momento. Además tiene un plugin, el ADT que es la herramienta de desarrollo de Eclipse para Android, y aporta un entorno de emulación configurable; aunque dadas las características de este proyecto se ejecutará el código directamente sobre el Smartphone, lo cual es posible gracias a la conexión que establece Eclipse con el móvil a través de adb que tiene un servidor que corre como un proceso en background que permite conectar el IDE al móvil y ejecutar las pruebas sobre él directamente. Entre otras muchas características estas son las principales por las que en entorno de desarrollo elegido es Eclipse.

- Etiquetas QR

Se crearán etiquetas QR en una web destinada a este fin de manera libre y se instalarán en electrodomésticos para su lectura. Poco queda por decir de estos códigos ya descritos en el apartado 2.4, sólo recordar que contienen información codificada accesible a través de un reconocedor que implemente alguna de las librerías creadas para este efecto.

- Librerías

  Para la implementación del proyecto se utilizarán 3 librerías:

  - zXing para el reconocimiento de QRs, ya que además de las propiedades descritas anteriormente, se programó y probó código utilizando la librería zBar; pero los resultados en cuanto a distancia fueron del todo inaceptables.

  - android-async-http-1.4.4 para gestionar peticiones post a cualquier servidor montado con independencia del utilizado en el proyecto.

  - quickblox-android-0.1.6 , librería utilizada para la gestión de la información en el servicio web utilizado, que es Quickblox.

- Quickblox

Quickblox es un servicio en la nube que aporta al desarrollador un entorno de backend para sus proyectos. Para utilizarlo es necesario darse de alta como desarrollador y registrar la aplicación que utilizará el servicio, tras lo cual se aportarán unas claves y token de autenticación que deberán estar configuradas en el código de la aplicación para establecer la conexión con el servidor. El desarrollador que actúa como administrador, puede crear tablas en la BBDD de Quickblox, acceder a la información almacenada y modificarla, entre otras muchas funcionalidades; ya que posee varios módulos de usabilidad, como la gestión de pantallas de registro de usuarios o el módulo utilizado para utilizar el servicio como backend de una aplicación. Existen modalidades de pago; pero básicamente para los módulos de chat y similares. Otro factor muy positivo es que si en algún momento se desea montar el backend en un servidor propio, es posible gracias a que Quickblox te facilita el código, aunque ya en este caso no de manera gratuita. Aunque para este proyecto sólo necesitamos la librería para Android, cabe destacar que provee librerías para otras plataformas, como iOS o incluso web, todas ellas gratuitas.

Figura 21. Credenciales de una app en Quickblox

2.8 Marco regulador

Descritas las tecnologías utilizadas vemos que de todas ellas se puede hacer un uso gratuito y libre sin problemas. Puesto que Android es libre, cualquiera puede subir una aplicación al Play Store (antiguo Market). La intención sería, una vez finalizado este proyecto e integrado con el sistema completo, subirla a mediante Google Play y que cualquier usuario de Smartphone con Android pudiera descargársela y utilizarla sin restricciones, nunca convirtiéndose en una aplicación, como muchas hoy, *In-App Purchase*

(en ellas la aplicación se ofrece al usuario de forma gratuita, pero luego para poder utilizarla al completo se debe pagar). Si en algún momento la aplicación necesitara monetizarse para poder seguir adelante con el proyecto, se podría hacer uso de la publicidad, incluso publicitando aplicaciones similares de otras empresas que ayuden a las personas con algún problema físico o de salud. Mención aparte merecen las políticas de privacidad, que en este caso no afectan directamente a la aplicación, ya que el usuario no necesita registrarse ni dar ningún dato, pero sí deberán tenerlas en cuenta todos aquellos fabricantes que implanten en su servidor las informaciones referidas a sus aparatos.

Volviendo al tema de publicar la aplicación en el Play Store existe una normativa marcada por el acuerdo y las políticas de distribución para desarrolladores, conjuntamente llamadas "Condiciones para Desarrolladores", citemos algunos de estos principios extraídos de la web de Google Play:

Políticas de contenido: No se permiten contenidos de violencia, apología de productos peligrosos ni actividades ilegales, entre otros muchos tipos de contenido nefasto. Además no es posible subir una aplicación al market que interfiera en el funcionamiento del sistema, como por ejemplo que realice cambios en el dispositivo del usuario fuera de la aplicación sin su conocimiento y consentimiento, o que incentive a eliminar o inhabilitar aplicaciones de terceros excepto si fuera parte de un servicio de seguridad proporcionado por la aplicación. Lo referido a la no instalación de otros recursos sin el consentimiento del usuario, es clave para este proyecto, ya que en principio la librería zXing preguntaba al usuario qué aplicación utilizar o descargar para el reconocimiento de QRs, pero de este modo se complicaba la navegación y por tanto se optó por hacer que la librería se integrara y ejecutara dentro de la propia aplicación desarrollada en este trabajo de fin de grado, para que funcione independientemente de otras que puedan no encontrarse en el terminal.

Spam y posicionamiento en Play Store: es muy importante ya que en base a ello, nuestra aplicación aparecerá antes o después en el listado de una búsqueda. Para lograr un buen lugar en la lista no se permite la publicación de contenido repetitivo, ni cambiar el posicionamiento manipulando las reseñas incluyendo valoraciones falsas entre otras, siendo claves para lograrlo una buena descripción, una selección inteligente de las palabras clave y un nombre adecuado de aplicación (características que se definen al subir la aplicación al Play Store, además de otros como imágenes de las pantallas).

Todo el conjunto de las condiciones de Google Play para desarrolladores deben ser cumplidas no sólo por la aplicación, sino también por las librerías externas utilizadas y/o los anuncios incluidos en la app o que estén disponibles a través de ella. Si alguna de las normas se incumpliera, Google Play podrá rechazar la aplicación tanto en el momento de subirla así como en una de sus revisiones y/o actualizaciones. Por ello, estas normas son de obligado cumplimiento para el desarrollo de cualquier aplicación Android.

Capítulo 3. Análisis, Diseño e Implementación

En las próximas páginas se expondrán las fases de análisis de requisitos, diseño de la aplicación e implementación del código, detallando el proceso seguido y justificando la toma de cada una de las decisiones. Al final se incluye un capítulo de pruebas realizadas para comprobar el funcionamiento de la aplicación.

3.1 Análisis

En este apartado se detalla el análisis de requisitos para la realización del proyecto. Las especificaciones se subdividirán en dos categorías: requisitos de usuario y de software; para posteriormente elaborar la matriz de trazabilidad. También incluye la definición de los casos de uso.

3.1.1 Requisitos de usuario

Este apartado detalla los requisitos de usuario subdivididos en dos categorías: requisitos de capacidad (¿qué necesita el usuario?) y restricciones (¿qué limitaciones hay?). Para facilitar la lectura los requisitos serán mostrados en tablas cuya nomenclatura está definida en el ANEXO II.

3.1.1.1 Requisitos de capacidad

APLICACIÓN MÓVIL			UR_C_01
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	El usuario debe poder manejar los electrodomésticos de su casa a través de la aplicación desarrollada ejecutable en un móvil.		

Tabla 2. Requisito de usuario UR_C_01

RECONOCIMIENTO DE ETIQUETAS QR			UR_C_02
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	El usuario debe poder reconocer una etiqueta QR situada en un electrodoméstico a través de la aplicación de un modo muy sencillo.		

Tabla 3. Requisito de usuario UR_C_02

VISUALIZACIÓN DE ETIQUETAS			UR_C_03
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	El usuario debe disponer de un terminal con cámara.		

Tabla 4. Requisito de usuario UR_C_03

OPCIONES DISPONIBLES			UR_C_04
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	Las opciones deben estar disponibles (remotamente) para cualquier usuario según fabricante y electrodoméstico, almacenadas en el servidor.		

Tabla 5. Requisito de usuario UR_C_04

SITUACIÓN DEL SISTEMA RECONOCEDOR			UR_C_05
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	La aplicación debe poder utilizarse desde una silla de ruedas, es decir debe ser una aplicación portable.		

Tabla 6. Requisito de usuario UR_C_05

VISUALIZACIÓN ACCIONES DISPONIBLES			UR_C_06
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	El usuario debe poder visualizar las opciones.		

Tabla 7. Requisito de usuario UR_C_06

AUDIO DE ACCIONES DISPONIBLES			UR_C_07
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	El usuario debe poder saber qué acciones tiene el dispositivo reconocido sin necesidad de ver la pantalla, escuchando el listado de las mismas.		

Tabla 8. Requisito de usuario UR_C_07

EMAIL SOPORTE TÉCNICO			UR_C_08
Prioridad	Media	Necesidad	Deseable
Estabilidad	Inestable	Verificabilidad	Alta
Descripción	El usuario debe poder contactar con el soporte técnico desde la aplicación.		

Tabla 9. Requisito de usuario UR_C_08

RECONOCIMIENTO DE ÓRDENES POR VOZ			UR_C_09
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	Con una sencilla orden por voz, el usuario debe poder ordenar una acción desde su móvil al electrodoméstico.		

Tabla 10. Requisito de usuario UR_C_09

RECONOCIMIENTO DE ÓRDENES POR PANTALLA			UR_C_10
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	El usuario debe poder manejar también la aplicación desde la pantalla del dispositivo.		

Tabla 11. Requisito de usuario UR_C_10

3.1.1.2 Requisitos de restricción

DISTANCIA DE RECONOCIMIENTO vs TAMAÑO QR			UR_R_11
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	Será imperativo que el móvil haga el reconocimiento de la etiqueta a una distancia superior a un metro, puesto que estará situado en la silla de ruedas y no podrá acercarse más a la etiqueta. Esto irá unido a la limitación de que la QR no deberá exceder los 9 cm de lado.		

Tabla 12. Requisito de usuario UR_R_11

ÓRDENES POR VOZ vs RUIDO			UR_R_12
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	Debe hacerse complicado que el ruido ambiental interceda en el funcionamiento de la aplicación.		

Tabla 13. Requisito de usuario UR_R_12

ÓRDENES RELATIVAS A UN ÚNICO DISPOSITIVO			UR_R_13
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	Las órdenes disponibles deben pertenecer sólo al electrodoméstico que acaba de ser reconocido.		

Tabla 14. Requisito de usuario UR_R_13

3.1.2 Requisitos de Software

Este apartado recoge los requisitos de software del proyecto, los cuales se dividirán en dos categorías: funcionales y no funcionales. De nuevo para facilitar la lectura se detallarán en tablas autoexplicativas cuya nomenclatura está definida en el ANEXO III. Debe tenerse en cuenta que los requisitos de software se establecen para garantizar el cumplimiento de los requisitos de usuario.

3.1.2.1 Requisitos funcionales

APLICACIÓN ANDROID			SR_F_01
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	Desarrollo de aplicación en Android para su instalación en terminales móviles.		
Dependencias UR	UR_C_01, UR_C_03, UR_C_05		

Tabla 15. Requisito de software SR_F_01

RECONOCIMIENTO EFICAZ DE LA ETIQUETA			SR_F_02
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	La aplicación se abre directamente en el modo de reconocimiento de etiquetas para facilitar el uso eliminando pantallas de transición innecesarias. Posteriormente todo el manejo se realizará en una misma pantalla realizando acciones sin pasos intermedios.		
Dependencias UR	UR_C_02		

Tabla 16.Requisito de software SR_F_02

TERMINAL SMARTPHONE			SR_F_03
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	El dispositivo será de pequeño tamaño, tendrá cámara, altavoz, micrófono y conexión a internet, características básicas de cualquier Smartphone.		
Dependencias UR	UR_C_03, UR_C_07, UR_C_08, UR_C_09		

Tabla 17. Requisito de software SR_F_03

SERVIDOR ALMACÉN DE OPCIONES			SR_F_04
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	Se trabajará con un servidor cuya función es almacenar las opciones disponibles según marca del fabricante del electrodoméstico y un identificador único.		
Dependencias UR	UR_C_04		

Tabla 18. Requisito de software SR_F_04

RECOGIDA DE ACCIONES DISPONIBLES			SR_F_05
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	La aplicación será capaz de hacer peticiones al servidor cuya respuesta sean las opciones disponibles para el aparato en cuestión.		
Dependencias UR	UR_C_04		

Tabla 19. Requisito de software SR_F_05

TERMINAL PORTABLE			SR_F_06
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	Será imprescindible para poder llevarlo en una silla de ruedas.		
Dependencias UR	UR_C_05		

Tabla 20. Requisito de software SR_F_06

VISUALIZACIÓN DE ACCIONES			SR_F_07
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	La pantalla principal mostrará al usuario, una vez reconocida una etiqueta, el listado de opciones disponibles.		
Dependencias UR	UR_C_06, UR_C_10		

Tabla 21. Requisito de software SR_F_07

LECTURA DE OPCIONES DISPONIBLES			SR_F_08
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	La aplicación reproducirá un audio con las opciones para que el usuario las escuche en caso de no poder leer bien la pantalla.		
Dependencias UR	UR_C_05, UR_C_07		

Tabla 22. Requisito de software SR_F_08

BOTÓN EMAIL SOPORTE			SR_F_09
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	La pantalla mostrará al usuario un botón (accionado por tap sobre él o mediante la voz) que permite enviar un email a la dirección del soporte técnico del electrodoméstico reconocido.		
Dependencias UR	UR_C_08		

Tabla 23. Requisito de software SR_F_09

INTEGRACIÓN GOOGLE VOICE SEARCH			SR_F_10
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	La aplicación tendrá integrado un sistema de reconocimiento de voz a través de Google Voice que llevan todos los dispositivos Android, para el reconocimiento de voz y ejecución de órdenes.		
Dependencias UR	UR_C_09		

Tabla 24. Requisito de software SR_F_10

SELECCIONAR UNA OPCIÓN			SR_F_11
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	El listado de opciones para un aparato es seleccionable mediante un tap sobre la acción deseada.		
Dependencias UR	UR_C_10		

Tabla 25. Requisito de software SR_F_11

3.1.2.2 Requisitos no funcionales

SELECCIÓN DE TIPO DE QR			SR_NF_12
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	La selección de la QR: de menor tamaño, con menor información contenida, con nivel de corrección de errores L (bajo), de alto contraste y situada en un entorno iluminado serán requisitos indispensables para una aceptable distancia de reconocimiento en nuestra aplicación.		
Dependencias UR	UR_R_11		

Tabla 26. Requisito de software SR_NF_12

ON/OFF RECONOCIMIENTO DE VOZ			SR_NF_13
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	El reconocimiento por voz se activará sólo en el momento en el que el usuario debe elegir qué hacer y nunca mientras otras opciones estén ejecutándose (cómo el reconocimiento de una etiqueta, el reconocimiento de una orden, o la lectura de las opciones) para impedir que el ruido de fondo o la voz de otras personas puedan interferir en el funcionamiento del sistema.		
Dependencias UR	UR_R_12		

Tabla 27. Requisito de software SR_NF_13

RECARGA DE PANTALLA CON CADA NUEVO RECONOCIMIENTO			SR_NF_14
Prioridad	Alta	Necesidad	Esencial
Estabilidad	Estable	Verificabilidad	Alta
Descripción	Cada vez que se reconozca una etiqueta, la pantalla sólo mostrará al usuario las opciones disponibles para el aparato en cuestión, en ningún caso se almacenarán posibles acciones de otros electrodomésticos.		
Dependencias UR	UR_R_13		

Tabla 28. Requisito de software SR_NF_14

3.1.3 Casos de uso

En este punto se presentan los casos de uso de la aplicación *DomoScan.* Bajo estas líneas podemos observar un diagrama explicativo del escenario de usabilidad.

Figura 22. Diagrama de casos de uso

- **Caso de Uso 1**

RECONOCIMIENTO DE QR	
Actor	Usuario de la aplicación.
Objetivo	Reconocer QR para obtener los datos pertenecientes al aparato elegido.
Pre-condición	- Conexión a internet - Conexión con el servidor que contacta con la BBDD - Enfocar la cámara hacia un código QR
Escenario	Usuario frente al aparato elegido para utilizar.
Post-condición	Se obtienen los datos de la QR y se envían al servidor para la obtención de la información almacenada en el mismo.

Tabla 29. Caso de Uso 1

- **Caso de Uso 2: Visualización y lectura de opciones disponibles**

VISUALIZACIÓN Y LECTURA DE OPCIONES DISPONIBLES	
Actor	La propia aplicación gestiona visualización y lectura de opciones.
Objetivo	Informar al usuario de las acciones que puede ejecutar en el aparato seleccionado.
Pre-condición	- Haber reconocido un QR - Activar la síntesis de texto a voz
Escenario	Usuario visualizando y escuchando las posibilidades de actuación.
Post-condición	Se muestran y escuchan las acciones disponibles para el usuario.

Tabla 30. Caso de Uso 2

- **Caso de Uso 3: Selección de acción a ejecutar**

SELECCIONAR ACCIÓN A EJECUTAR	
Actor	El usuario de la aplicación.
Objetivo	Mandar una orden al teléfono, reconocerla y enviarla al electrodoméstico.
Pre-condición	- Conexión a Internet - Disponer del sistema de reconocimiento Google Voice - Activar reconocimiento de voz
Escenario	El usuario emite una orden por voz y el teléfono la reconoce.
Post-condición	Se muestra un mensaje indicando qué acción ha sido reconocida si está permitida y si no se muestra un mensaje de error.*

Tabla 31. Caso de Uso 3

*Nota: el objetivo de este proyecto es la parte de un sistema completo de domótica, que consiste en desarrollar la parte de reconocimiento de imágenes y voz en la aplicación móvil; por lo que no se implementa la conexión con el propio electrodoméstico; sino que ésta será representada por el mencionado mensaje de aceptación.

Estos 3 son los casos de uso principales, dada la limitación de páginas, los cuadros explicativos de los casos de uso secundarios, se pueden ver en el Anexo IV.

3.1.4 Matriz de trazabilidad

Durante el tiempo que se lleva a cabo un proyecto es muy importante tener claros los requerimientos del mismo. Así nace la matriz de trazabilidad, la cual relaciona de un modo sencillo los requisitos de usuario con los requisitos de software. Para que el proyecto esté terminado correctamente deben cumplirse todos los requisitos de usuario, lo que significa que en la matriz de trazabilidad debe comprobarse que cada requisito de usuario esté contemplado por uno o varios requisitos de software. En los proyectos grandes, el Project Manager utiliza la matriz para asegurar que se cumplen todos los requisitos de manera eficaz durante el ciclo de vida del proyecto, para lo cual realiza un seguimiento a toda la información y analiza las especificaciones cuando haya cambios propuestos al alcance del proyecto. (13)

Las directrices para construir la matriz son sencillas:

- El eje vertical de la tabla contiene los identificadores de requisitos de usuario (de capacidad y restricción siguiendo la nomenclatura descrita en el ANEXO II).
- Por el contrario, en el eje horizontal se encuentran los identificadores de requisitos de software.

Como hemos visto en la descripción de requisitos de software, cada uno de ellos llevaba asociado uno o varios requisitos de usuario en el campo *Dependencias UR,* por lo que a partir de ellos podemos construir la matriz de trazabilidad:

	SR_F_01	SR_F_02	SR_F_03	SR_F_04	SR_F_05	SR_F_06	SR_F_07	SR_F_08	SR_F_09	SR_F_10	SR_F_11	SR_NF_12	SR_NF_13	SR_NF_14
UR_C_01	X													
UR_C_02		X												
UR_C_03	X		X											
UR_C_04				X	X									
UR_C_05	X					X		X						
UR_C_06							X							
UR_C_07			X					X						
UR_C_08			X						X					
UR_C_09			X							X				
UR_C_10							X				X			
UR_R_11												X		
UR_R_12													X	
UR_R_13														X

Tabla 32. Matriz de trazabilidad: Requisitos de usuario – Requisitos de software

- ✓ Para comprobar que todos los requisitos de usuario quedan cubiertos es preciso verificar que ninguna fila queda vacía. Como este es el caso de la matriz que acabamos de analizar, podemos asegurar que todas las especificaciones serán implementadas en código, si no fuera así, sería necesario realizar un nuevo análisis del problema proyecto antes de realizar el diseño del proyecto.

3.2 Diseño

El presente apartado describe el diseño llevado a cabo para la realización del proyecto. Se detallarán la arquitectura del sistema, el modelo vista controlador empleado, el diseño de la base de datos y de la interfaz y se presentará un diagrama de flujo a fin de exponer claramente el funcionamiento de la aplicación *DomoScan*.

3.2.1 Arquitectura del sistema

En el proceso de diseño de la arquitectura del sistema se valoraron varias opciones, que es preciso explicar para entender cómo se llega a plantear el diseño elegido.

- **Etiqueta QR con toda la información contenida**

La forma más sencilla de implementar el sistema era que la propia QR de cada aparato almacenara todas las opciones disponibles. Pero de esta manera no se conseguía

la premisa fundamental, la distancia de reconocimiento. Es fundamental jugar con las versiones de las QR para manejar tamaños de etiquetas y distancias de reconocimiento aceptables para el entorno de usabilidad, en el que la etiqueta no puede superar los 10 cm de lado y la distancia de reconocimiento ha de ser 1m como mínimo. Si incluíamos toda la información en la QR, el proyecto incumplía esta condición básica por lo que rápidamente fue descartada.

- **Etiqueta QR con sólo parte de la información contenida**

Descartada la primera opción, llegamos a plantear que sólo parte de la información estuviera contenida en la QR, es decir, que contuviera la referencia a un servidor de datos que almacenara parte del contenido mientras que el resto de opciones aparecieran reflejadas en la QR, como las acciones más habituales de un electrodoméstico. Tras comprobar que las conexiones con el servidor y la descarga de los datos no tenían prácticamente retardo, era innecesario almacenar parte de la información en la QR y que esta tuviera que aumentar de tamaño para mantener una distancia de reconocimiento aceptable, ya que al contener más datos la versión no podría ser v1 y por tanto costaría más su reconocimiento.

- **BBDD en local con todas las etiquetas almacenadas**

También se consideró la idea de tener una BBDD con el contenido y que la QR almacenara la referencia a la misma; pero es directo darse cuenta de los problemas de escalabilidad que presentaba esta opción. Cada vez que un nuevo proveedor de aparatos sacara un nuevo producto o un usuario se comprara uno nuevo, la aplicación móvil debía actualizar la BBDD en el Smartphone, algo totalmente ineficiente ya que entonces la aplicación debía cambiar para adaptarse a cualquier cambio.

- **Etiqueta QR con referencia al servidor de contenidos**

Seleccionando lo mejor de cada una de las opciones anteriores llegamos al diseño final: un servidor de contenidos accesible por el usuario a través de la aplicación *DomoScan* instalada en su móvil. Para el desarrollo de este proyecto se ha utilizado el servicio en la nube Quickblox que provee servicio de BBDD y su propia librería para hacer las peticiones al servidor y obtener los datos. No obstante cualquier proveedor pueda crear sus etiquetas QR y tener su BBDD en un servidor propio y la QR sólo deberá contener la url con los parámetros de petición al mismo, ya que la aplicación al detectar una url hará la petición http para obtener los datos. Por lo tanto la información que poseerán las QR podrá ser de 2 tipos:

- Una url con la petición a un servidor propio de una empresa externa que no quiera utilizar el servicio Quickblox.

- Un identificador numérico y la marca del electrodoméstico que son los parámetros con los que se hará la petición al servidor Quickblox para obtener la información relativa a la QR reconocida.

Además del servidor de contenidos, es importante señalar que otro servidor entrará en la arquitectura para gestionar el reconocimiento de voz y devolver la orden reconocida tras el habla del usuario, estos son los servidores de Google, los mismos que utiliza el Smartphone cuando presionamos el botón del micrófono que se muestra al lado del teclado, o cuando entramos en el widget del buscador.

Por todo ello, vemos que la arquitectura del proyecto tiene una clara estructura Cliente-Servidor, que es una arquitectura distribuida en la que una o varias aplicaciones cliente hacen peticiones a un servidor, que a su vez da una respuesta (con los datos solicitados por el usuario en la aplicación) tras recuperar la información de una BBDD. Siempre se partirá de una acción por parte del usuario, que desencadenará la orden al Smartphone que a su vez lanzará la petición al servidor, el cual responderá; y en ningún caso al revés. Veámoslo en la siguiente figura:

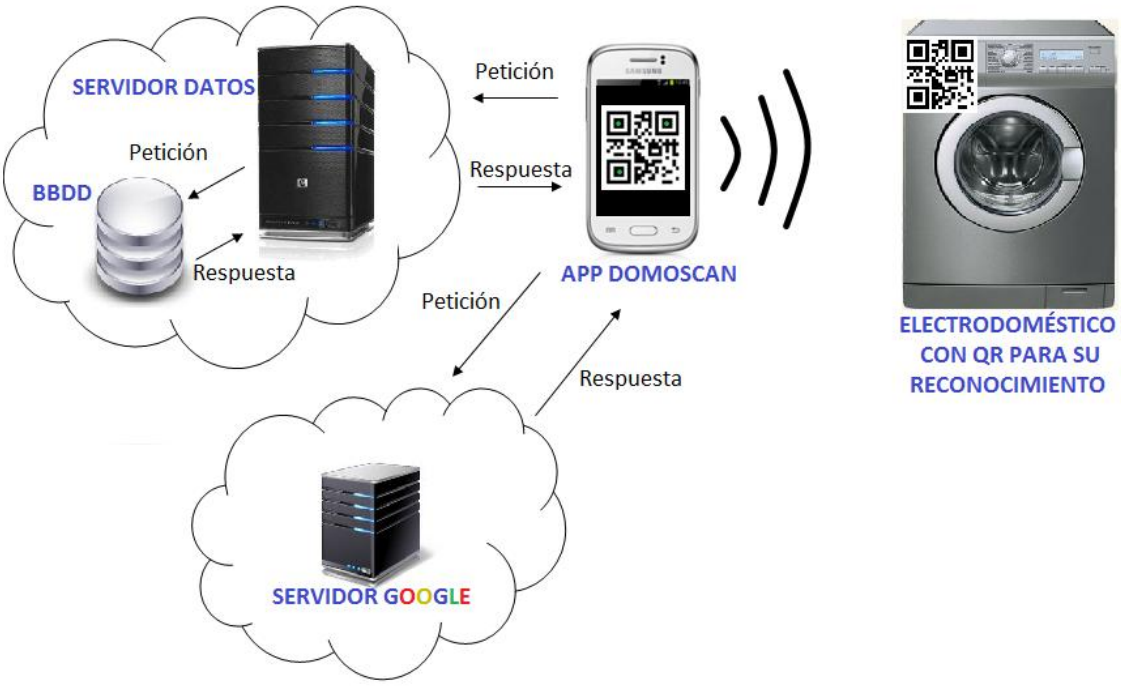


Figura 23. Arquitectura de la aplicación *DomoScan*

Dada esta arquitectura existen 3 niveles claramente diferenciados:

- Nivel 1: que corresponde con el dispositivo que posee la aplicación DomoScan

- Nivel 2: servidor de datos para gestionar las peticiones

- Nivel 3: base de datos que almacena los modelos de información

Así, el proyecto tiene las 3 capas funcionales claramente diferenciadas, lo que nos permite diseñarlo siguiendo el modelo de vista controlador (MVC).

3.2.2 Modelo Vista Controlador

El MVC es un patrón de arquitectura de software que separa los datos y la lógica de una aplicación de la interfaz de usuario. Se basa en la ideas de reutilización de código y separación de conceptos que facilitan el desarrollo y mantenimiento de las aplicaciones. En este patrón hay 3 componentes:

- el Modelo: es el conjunto de clases que representan la información del mundo real que el sistema debe reflejar.

- la Vista: es la encargada de la representación visual del modelo al usuario.

- el Controlador: recibe y responde a los eventos, invoca peticiones al modelo, hace el tratamiento de la información, es decir sirve de intermediario entre la vista y el modelo.

El flujo del MVC que sigue el proyecto sería el siguiente:

1º. Se lanza un evento a través de la interfaz, mediante una orden por voz o un tap en la pantalla.

2º. El controlador recibe y gestiona la petición.

3º. El controlador envía la respuesta a la vista manejando el modelo de datos.

4º. La vista muestra el resultado que le indica el controlador y queda a la espera de otro evento.

La correspondencia de los 3 elementos del MVC con el proyecto sería la siguiente:

- el Modelo corresponde con las clases que se encargarán de la modelización de los datos y por lo tanto irá ligado al diseño de la base de datos.

- la Vista es muy sencilla de identificar en un proyecto desarrollado en Android, puesto que la propia plataforma ya hace una separación entre la lógica y la interfaz, ya que la manera de desarrollar las vistas es mediante los fichero XML que contiene la carpeta layout de un proyecto Android. Cada XML corresponderá con la interfaz de cada pantalla.

- el Controlador se identifica con el resto de clases, tanto las activities como aquellas encargadas de la conexión con el servidor, los listeners de los eventos, los

adapters de las listas, etc. En definitiva, cualquier clase que gestione datos, peticiones y respuestas.

3.2.3 Diseño de la Base de Datos

Partiendo de la base de que el usuario deberá poder tener en su móvil de todas las acciones disponibles para un electrodoméstico con tan sólo reconocer un código, es necesario que dichas acciones estén almacenadas y relacionadas con el aparato en cuestión de un modo remoto, para que cualquier usuario con el mismo electrodoméstico tenga acceso a las mismas funcionalidades. Para ello se ha utilizado el servicio en la nube Quickblox que provee de un módulo, llamado 'CustomObjects'. Con este servicio, el administrador puede, tan sólo logándose en la web, introducir datos para su almacenamiento persistente en los servidores MySQL que posee Quickblox, para luego a través de la librería del mismo nombre, poder acceder a ellos con una sencilla petición al servidor, que a su vez hace la consulta a la base de datos, que contiene objetos de cada modelo.

El diseño de la base de datos se ha realizado partiendo de la modelización pensada para la estructuración de la información. La idea es tener 2 modelos: 'Appliances' y 'Options' que dan lugar a 2 tablas en la base de datos. Bajo estas líneas, describimos ambas:

- **Tabla 'Appliances'**

Será la encargada de almacenar toda la información relacionada con un electrodoméstico, a través de un identificador numérico único y el nombre de la marca. Al dar los dos parámetros se permite a cada marca gestionar sus identificadores con independencia del resto de fabricantes y relacionar un aparato con sus características de manera inequívoca. Las columnas de esta tabla así como su descripción se detallan a continuación:

- deviceID: identificador único de un electrodoméstico.
- name: nombre genérico del aparato.
- serialNumber: número de serie asignado al electrodoméstico.
- imageQR: etiqueta QR en formato jpeg para dejar registrada la etiqueta que se distribuirá con el aparato.
- brand: marca del electrodoméstico.
- contactUs: contendrá el email de contacto del soporte técnico del fabricante.
- actions: listado de identificadores de acciones disponibles para este electrodoméstico.
- Created_at: fecha y hora en la que se creó/modificó el registro de un electrodoméstico.

La siguiente imagen muestra la tabla descrita:

Class Appliances ▾ Show 10 ▾ entries

imageQR	name	serialNumber	brand	contactUs	deviceID	actions	Created at
Bosch1.jpe	Lavadora Z1	5681239425669784	Bosch	tech_support@bosch.co...	1	[1, 2, 3, 4, 5, 6]	2014-04-18 18:40:59 UTC
Liebherr2.jp	Frigorifico LeT4	84631297630474	Liebherr	tech_support@liebherr...	2	[7, 8, 9, 10, 11, 12,...	2014-04-18 18:45:41 UTC
Balay3.jpeg	Horno Px9	896547236519675300	Balay	tech_support@balay.co...	3	[1, 17, 18, 19, 20, 2...	2014-04-18 18:51:35 UTC

Figura 24. Tabla 'Appliances' de la BBDD

- **Tabla 'Options'**

Esta estructura de datos será la responsable de relacionar cada identificador de acción con la propia acción, de manera que varios electrodomésticos puedan hacer uso de una misma acción sin necesidad de repetirla, sino únicamente utilizando el mismo identificador. Así por ejemplo para la acción de 'abrir' una lavadora y un lavavajillas tenemos una sola acción que se asignará a ambos aparatos. Pasamos a definir las 2 columnas que tendrá esta tabla:

- optionID: identificador único de acción.
- action: nombre de la propia acción.
- Created_at: fecha y hora de la creación/modificación del registro, a fin de saber cuándo se produjo la última actualización de los datos.

Aunque dos marcas pudieran coincidir en el mismo optionID no llevaría a equívoco ya que la tabla 'Appliances' contiene el nombre de la marca y por lo tanto no sería lo mismo un identificador de marcaX con uno de marcaY; pero tal y como se especifica en la descripción de la columna optionID, éste sí debe ser único dentro de una misma marca.

A continuación se muestra la tabla 'Options' con algunos registros:

Class Options ▾ Show 10 ▾ entries

optionID	action	Created at
1	Apagar	2014-04-19 12:44:59 UTC
2	Pausar	2014-04-19 12:45:09 UTC
3	Seleccionar programa ...	2014-04-19 12:45:22 UTC
4	Seleccionar programa ...	2014-04-19 12:45:37 UTC
5	Iniciar lavado	2014-04-19 12:45:54 UTC
6	Lavado en frío	2014-04-19 12:46:18 UTC
7	Bajar temperatura fri...	2014-04-19 12:47:13 UTC
8	Subir temperatura fri...	2014-04-19 12:47:24 UTC
9	Bajar temperatura con...	2014-04-19 12:56:55 UTC
10	Subir temperatura con...	2014-04-19 12:57:10 UTC

Figura 25. Tabla 'Options' de la BBDD

3.2.4 Interfaz de usuario

Dadas la discapacidad del usuario final, uno de los objetivos fundamentales de este proyecto era hacer una interfaz muy sencilla, que presentara las opciones al usuario de una manera clara y le permitiera interactuar con ella fácilmente. La idea de varias pantallas de navegación complicaría el objetivo además de ser totalmente innecesarias. Por lo tanto el diseño constará tan sólo de una pantalla de carga (la splash), tras la cual que mostrará la pantalla de escaneo de QR y una vez reconocido se irá a la vista principal para gestionar los eventos:

- **Splash**
  La splash es la pantalla que típicamente sólo muestra el logotipo y nombre de la aplicación y una barra de progreso. Se utiliza para informar al usuario de que la aplicación se está abriendo mientras se cargan las librerías y se establecen los parámetros y conexiones necesarios para el funcionamiento de la aplicación; tales como verificar la conexión a internet, realizar las conexiones con servidor, el registro de la aplicación en el mismo (requerido para utilizar Quickblox).

Figura 26. Pantalla Splash

- **Pantalla de reconocimiento QR**
  La pantalla de reconocimiento no tiene ningún diseño pues únicamente se abre la cámara del Smartphone para que el usuario enfoque a la QR, situándose delante de ella y se reconozca el electrodoméstico elegido. En la pantalla se muestran: una línea roja vertical que divide la pantalla en 2 mitades y que sirve para ayudar al usuario a enfocar, y puntos verdes que se sitúan por la pantalla en los puntos donde la librería zXing reconoce datos para finalmente fijarse en los 3 patrones del QR (momento en el que el reconocedor ha detectado y leído la etiqueta).

Figura 27. Pantalla de reconocimiento

- **Pantalla principal**
  Es la pantalla en la que el usuario podrá lanzar eventos mediante el reconocimiento de voz. Se divide en 3 secciones:
  - la primera contiene el botón de escaneo, siempre disponible para que el usuario pueda escanear un electrodoméstico cuando desee.

- una segunda en la que se muestra el nombre genérico del electrodoméstico reconocido (o indica al usuario que escanee uno en caso de no haberlo hecho).
- y la tercera es en la que, en caso de haber escaneado una QR, se muestra un botón para contactar con el servicio técnico del aparato reconocido y un listado de acciones correspondientes al mismo. Tanto los botones como el listado se activan mediante el reconocimiento de voz (también tocando sobre ellos en la pantalla).

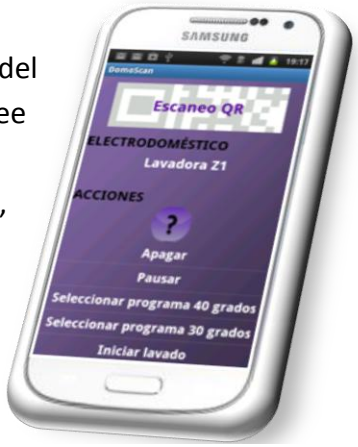


Figura 28. Pantalla principal

Además de las pantallas propias de la aplicación, se utilizarán también las pantallas de redacción de email, al contactar con el servicio técnico (botón con interrogación) y los diálogos mostrados para el reconocimiento de voz.

Figura 30. Pantalla de aplicación de correo

Figura 29. Reconocimiento de voz

Sin embargo la premisa en cuanto a sencillez de manejo de las pantallas descrita no es vinculante a la hora de hacer el diseño gráfico de botones, fondos, títulos, listados, etc. En este aspecto el usuario es un usuario más, al que satisfacer con un diseño atractivo y original. Por ello se ha optado por intentar hacer un diseño significativo, en el que cada grafismo tuviera su simbología y originalidad. Hablamos ahora un poco de este aspecto:

- **Icono**

Con el diseño del icono de la aplicación se ha intentado reivindicar la idea en la que se basa el proyecto, el facilitar determinadas acciones a personas con problemas de movilidad. Por ello la QR del icono, que se ha creado para el mismo, contiene una cadena de texto que dice: 'Rompe las barreras' y se superpone encima del dibujo de una barrera, a fin de simbolizar la pequeña ruptura con los problemas de accesibilidad que puede aportar el sistema completo en el que se integra la aplicación.

Figura 31. Icono *DomoScan*

- **Título**

El título *DomoScan* es el resultado de mezclar las palabras de las que parte la creación de este proyecto: domótica y escaneo de

Figura 32. Logo *DomoScan*

etiquetas. En la splash, el título ha sido diseñado con barras horizontales, emulando un escáner, lo cual ha sido posible gracias a un servicio online destinado al diseño (14). Incluye también un pequeño dibujo que simula el objetivo de una cámara.

- **Botones 'Escaneo QR' y soporte técnico**

El botón de soporte técnico se ha diseñado en una página web destinada a esta causa (15), para darle sombreado y distintos tonos, además de la forma redondeada. El botón de escaneo QR tiene como fondo una parte del código QR creado para el logotipo, que se oscurece al activarlo. También el botón de soporte tiene la animación de cambiar de color al ser seleccionado, a fin de dar la sensación de pulsación al usuario.

- **Fondo, colores y letras**

El fondo de las pantallas es sencillo para no entorpecer la legibilidad del contenido y los colores son llamativos para que no sea una aplicación 'aburrida' a ojos del usuario. Las letras y el contraste con el fondo siguen la misma idea de diseño.

3.2.5 Diagrama de flujo

Para facilitar la comprensión del funcionamiento de la aplicación al lector se expone a continuación el diagrama de flujo de la misma. Es necesario hacer algunas aclaraciones sobre la figura que mostraremos:

- Se entenderá que la palabra acciones se utiliza para hacer referencia a las acciones propias de la aplicación (cerrar aplicación, escanear código QR) mientras que el término opciones se referirá a las opciones características de un aparato reconocido (abrir lavadora, encender microondas, etc).
- Dado que este proyecto versa de la aplicación para el desarrollo de un sistema integrado con domótica, cuando se habla de un mensaje en pantalla que muestra la opción seleccionada, nos referimos a un toast que aparecerá por un periodo de tiempo, simulando el lanzamiento de la opción sobre el electrodoméstico elegido.
- Como se ha comentado con anterioridad, la aplicación también es manejable mediante los eventos táctiles sobre la pantalla, por lo que el siguiente diagrama de flujo también representa este otro tipo de navegabilidad. Únicamente variaría que en vez del reconocimiento de voz, se mantendría a la espera de un tap sobre la pantalla.

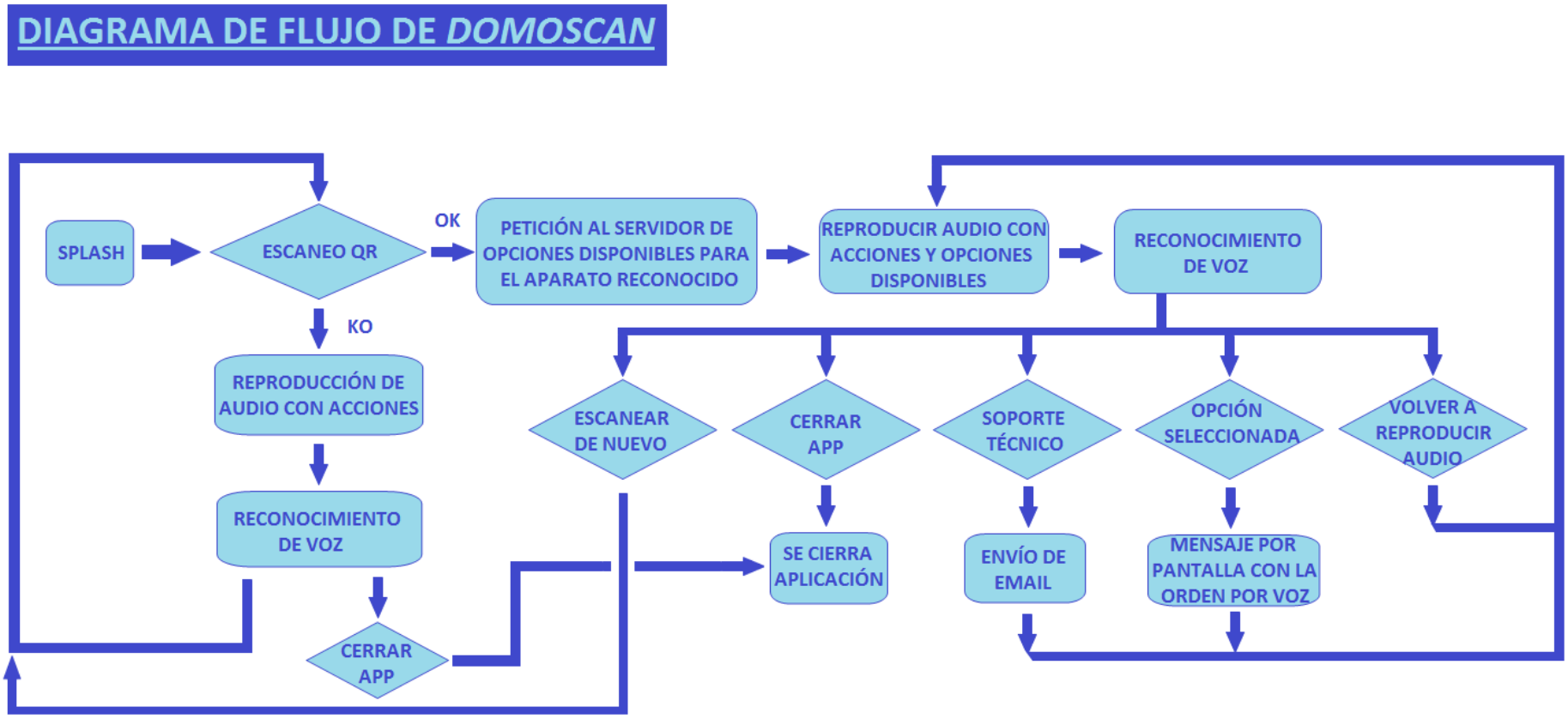


Figura 33. Diagrama de flujo de *DomoScan*

3.2.6 Diagrama de clases

Todo lo explicado anteriormente nos lleva a diseñar el siguiente diagrama de clases que implementen el flujo y con las funcionalidades requeridas. Dichas clases serán detalladas en el siguiente apartado.

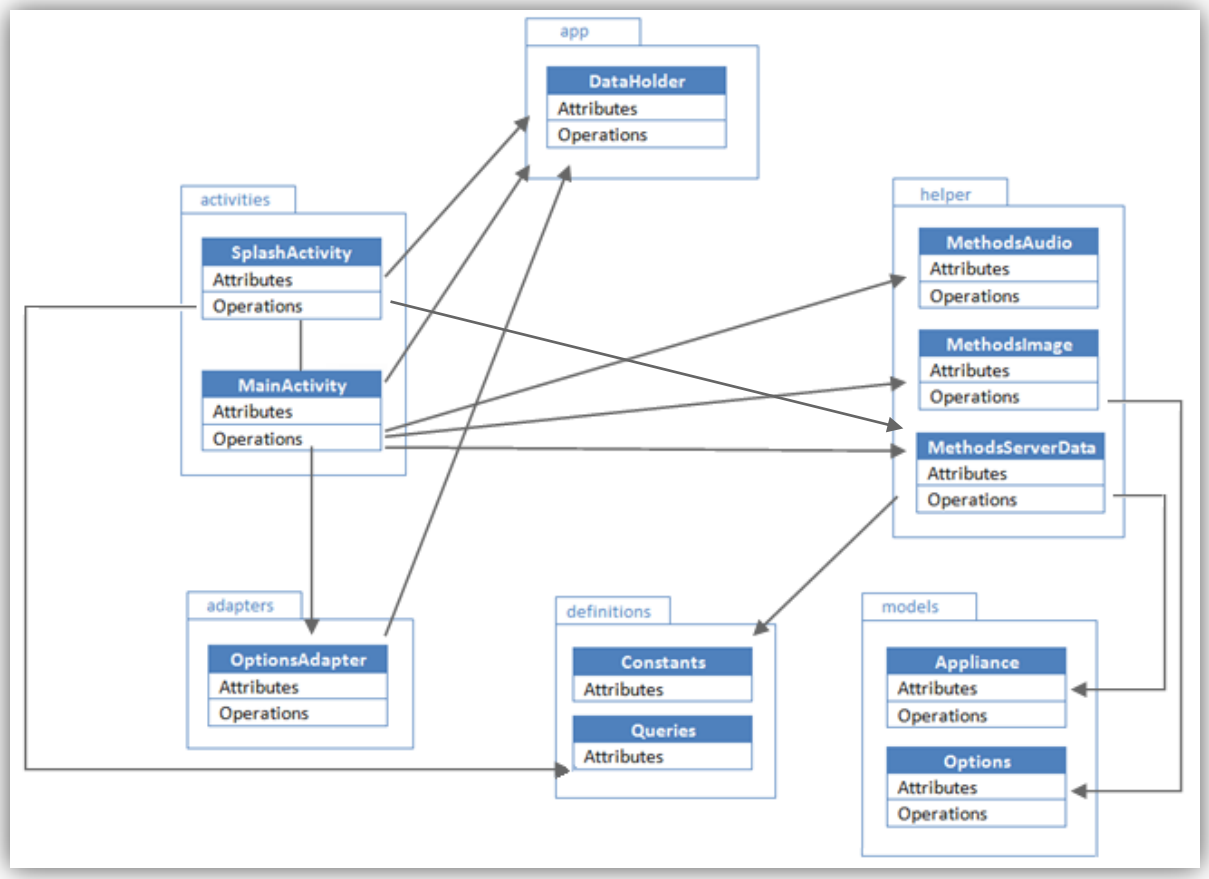


Figura 34. Diagrama de clases

3.3 Implementación

Para el desarrollo de este proyecto han sido utilizados dos tipos de patrones de diseño: el MVC, comentado en el capítulo anterior, y una clase singleton. En la siguiente figura podemos ver los diferentes paquetes que estructuran el proyecto agrupando las clases por funcionalidad y pertenencia a un componente del patrón.

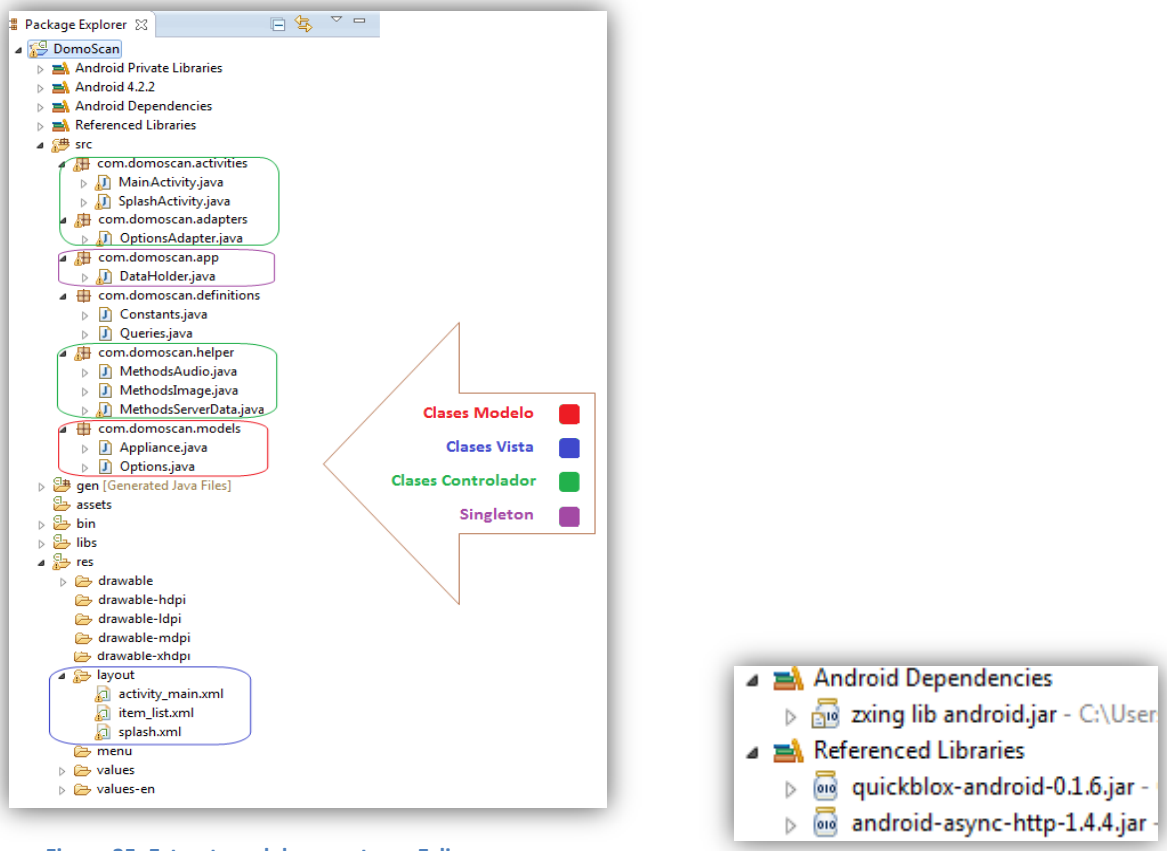


Figura 35. Estructura del proyecto en Eclipse

Además, en la imagen a la derecha se observan las librerías utilizadas, que son:

- *zxing-lib-android.jar* para el reconocimiento de códigos ,
- *Android-async-http-1.4.4* para peticiones post a un servidor externo como explicaremos más adelante,
- y *quickblox-android-0.1.6.jar,* una librería propia del servicio en la nube utilizado para desarrollar la parte del servidor y comunicaciones con la BBDD, que proveerá métodos para gestionar la conectividad con el servidor, registro en el servicio y realizar las peticiones de datos.

A continuación vamos a comentar:

- la implementación de la interfaz
- los aspectos más relevantes de cada clase, (agrupadas en paquetes) atributos y métodos, y su interacción con el resto,
- fichero manifest
- y finalmente se presentará el diagrama de clases que representará lo explicado.

3.3.1 Interfaz

El primer paso en la realización de una aplicación móvil es hacer un diseño de la interfaz, para poder decidir cómo será el flujo de navegación y a partir de ahí poder desarrollar la lógica de funcionalidades. La plataforma Android en sí misma hace que el desarrollo de la interfaz vaya en ficheros XML, por lo que estética y funcionalidad quedan separados, lo cual nos permite fácilmente aplicar el modelo MVC: los XML serán la vista, es decir la apariencia de las pantallas que se mostrarán al usuario. Así nuestro código se presentará mucho más ordenado y claro, utilizando los archivos XML para decidir tamaños, formas, colores, ubicación, incluso estilos; y los ficheros java para la lógica de la aplicación.

Los ficheros XML se deben almacenar en el directorio /res/layout de nuestro proyecto, tal y como vimos en la figura mostrada en la página anterior.

Para este proyecto, el objetivo era hacer una aplicación sencilla y rápida de manejar, por ello como ya vimos en el diseño, la aplicación consta únicamente de una pantalla de carga llamada splash, y la pantalla principal desde la que el usuario podrá realizar cualquier acción. Si nos fijamos en la imagen de la estructura del proyecto, vemos que aparece otro XML, item_list.xml. El motivo es que la pantalla principal muestra el listado de opciones disponibles para un electrodoméstico reconocido en un listado, pues bien, ítem_list.xml define el formato de cada fila de dicho listado.

Finalmente, para que el diseño creado en el fichero XML se presente como interfaz de una *Activity* (pantalla) hay que establecerlo como vista de dicha clase, de la siguiente manera:

```
public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
```

Figura 36. Asignación de vista a la Activity

3.3.2 Paquete *app*

En este paquete tenemos la clase que podría definirse como el núcleo de los datos de la aplicación. Está diseñada con dos características fundamentales: es un singleton y hereda de la clase 'Application' de Android. Dicha herencia significa que nuestra clase App contendrá los datos que deseemos durante todo el ciclo de vida de la aplicación, sin necesidad de pasarlos entre activities o utilizar una BBDD local,(lo cual sería poco recomendable para almacenar una cantidad pequeña de información).

Además, todo lo que contenga la clase, será ejecutado al lanzar la aplicación, para lo cual sólo es necesario declararla en el manifest de la siguiente manera:

```
<application
    android:name="com.domoscan.app.App"
```

Figura 37. Asignar clase *application* a un proyecto

Así, Android la reconoce al abrir la app y ejecuta los métodos que le indiquemos. Para optimizar este recurso, la declaramos además como singleton. Una clase definida como singleton tiene la característica de que sólo será instanciada una vez, pudiendo modificar el valor de los atributos las veces que se quieran; pero sin necesidad de crear objetos constantemente, ocupando memoria de manera innecesaria. Siguiendo un modelo UML, esta es su estructura:

App
Atributos
`private static AppdataHolder;` `private HashMap<Integer,String>optionsList;`
Métodos

`public App()`
`public static synchronized App getInstance()`
`public HashMap<Integer,String>getListOptions()`
`public void setListOptions(HashMap<Integer,String>arrayDado)`
`public int getOptionsListSize()`
`public ArrayList<String>getOptions()`
`public void clear()`

Tabla 33. UML clase App.java

La clase App tendrá la información obtenida de un electrodoméstico tras el escaneo del QR, es decir, fundamentalmente almacenará en el atributo *HashMap* las opciones disponibles junto a su identificador único. Los métodos que tiene gestionarán esta información, pudiendo acceder a ella, obtener el tamaño del listado, etc.

Mención especial merecen los métodos constructor y *getInstance()*. Dentro de este último se llamará al constructor de la clase, por si aún no se ha creado el objeto único de la clase App, hacerlo ahí. Este método tiene dos modificadores importantes: *static*, lo que hará que se pueda acceder al método referenciándolo desde fuera directamente con la clase App, y *synchronized*, el cual es el que aporta la singularidad de única instancia, propia del singleton como comentábamos antes.

Este tipo de clases que heredan de 'Aplication' se referencian de la siguiente manera:

```
App a;
a=(App)getApplication();
a.getListOptions().clear();
```

Figura 38. Uso clase *application*

3.3.3 Paquete *activities*

Las dos clases de este paquete extienden de la clase *Activity* que es el componente que representa una pantalla del terminal móvil.

- En primer lugar tenemos la *SplashActivity*. La Splash es una pantalla típica en prácticamente todas las aplicaciones del mercado que se utiliza para fines comerciales, presentando publicidad de la empresa; pero que podemos hacerla muy útil empleándola para realizar conexiones, registros, etc., lo cual agiliza las posteriores pantallas ya que las acciones generales que suponen una espera ya se han cargado. En el proyecto, no se ha utilizado con fines comerciales, sino para establecer el registro necesario de la aplicación en el servicio en la nube Quickblox. Su estructura es la siguiente:

SplashActivity

Atributos
`private ProgressBarprogressBar;`
`private final String TAG="SPLASH";`

Métodos
`protected void onCreate(Bundle savedInstanceState)`
`public void onComplete(Result result)`
`public void onComplete(Result result, Object context)`
`private void startMainActivity()`

Tabla 34. UML clase SplashActivity.java

Como podemos ver sólo tenemos el TAG empleado para mostrar logs y una *ProgressBar* como atributo, para mostrarla al inicio de la actividad y hacerla desaparecer cuando en el método *onComplete()* registremos la finalización correcta de las acciones lanzadas en el *onCreate()* que fundamentalmente, además de cargar la vista, serán la de registrar la aplicación con el método correspondiente de la clase MethodsServer (*registerApp()*).

Como podemos ver son dos métodos de la librería de Quickblox que hacen las dos acciones básicas para utilizar el servicio: en primer lugar se dan nuestras credenciales obtenidas tras registrarnos como desarrolladores y dar de alta la aplicación y después se envía la petición de autorización para conectar con el mismo, tras lo cual se recibe un token de autorización y la aplicación ya puede utilizar todos los recursos habilitados por el administrador, en nuestro caso el módulo *"CustomObjects"* que sirve para almacenar objetos del modelo de datos en el servidor.

La Splash presenta un diseño gráfico sin interactividad con el usuario (grafismo que vimos en el capítulo 3.2.4 interfaz de usuario), con una barra de progreso, tras la cual pasa a la siguiente pantalla: la MainActivity.

- La MainActivity es, como ya dijimos, la pantalla principal y única de interacción con el usuario (además de la pantalla que habilita la cámara para el reconocimiento de QRs que es propia de la librería zXing).

MainActivity

Atributos

```
private final String TAG="MAIN";
private TextView tvStatus;
private ListView listaOp;
private Button contactBtn;
private Button scanBtn;
private Context cx;
private OptionsAdapter adapter;
private ArrayList<String>optionsList=new ArrayList<String>();
private HashMap<Integer,String>hashOptions=new HashMap<Integer,String>();
private Appliance appli;
private Options opt;
private Handler mHandler = new Handler();
private static final int RECOGNIZE_SPEECH_ACTIVITY = 1;
private TextToSpeech textToSpeech;
private String textoReproducir;
```

Métodos

```
protected void onCreate(Bundle savedInstanceState)
public void onActivityResult(intrequestCode, intresultCode, Intent intent)
public void onComplete(Result result)
public void onItemClick(AdapterView<?> a, View v, int position, long id)
public void onClick(View v)
public void onInit(int status)
public void onUtteranceCompleted(final String utteranceId)
public void onBackPressed()
protected void onDestroy()
```

Tabla 35. UML clase MainActivity.java

En los atributos tenemos además del TAG para los *logs*, los propios de las vistas, como los botones, el listado para las opciones y su adapter, el *textview* para mostrar el nombre del electrodoméstico reconocido, el contexto, el listado de opciones y un hash que lo relaciona con su identificador de acción, un objeto de cada modelo, un hadler para manejar hilos de ejecución (como hilos de espera a terminar acciones) y los últimos atributos que son para la reproducción del audio con las acciones encontradas.

Para hablar de los métodos, lo haremos siguiendo el flujo de la actividad. En primera instancia se llama al método *onCreate()* en el que se asignan las vistas (tanto al *layout* como a la *listview* que contiene), el *adapter*, los *listeners* de eventos para los botones y la lista, se llama a los métodos para configurar el *Text2Speech* y se lanza directamente el reconocimiento de QR, con el método *startActivityForResult()*y la configuración de los parámetros para el reconocimiento de la clase *MethodsImage*. Tras el reconocimiento se ejecuta el método *onActivityResult()* que recoge el resultado de la lectura de la QR, hace las llamadas a los métodos de la clase *MethodsServer* para la petición de datos (según si hacemos la petición al servidor utilizado Quickblox o a un servidor externo), tipifica como objetos del modelo la respuesta, obtiene el listado de opciones y lo notifica al adapter de la lista. Estas peticiones asíncronas, llamarán internamente al método *onComplete()* cuando concluyan. Una vez obtenida la

respuesta, se lanza el *Text2Spech*(configurando parámetros e intent con los métodos de la clase *MethodsAudio*) para la escucha de las opciones disponibles, entre las que están además de las propias del aparato, la de cerrar la aplicación o volver a reconocer etiqueta entre otras. Puesto que la clase además de heredar de *Activity*, implementa la interfaz *TextToSpeech.OnInitListener,* al iniciar la reproducción del audio se llamará al método *onInit()* el que controla que esté soportada la reproducción y establece un escuchador del evento fin de la reproducción, momento en el que el método *onUtteranceCompleted()* es llamado. Aquí se encuentra la llamada a *Google Voice* para hacer el reconocimiento de voz, tras lo cual se convierten las palabras en un *ArrayList* de *String*en el método *onActivityResult()* y se ejecuta un evento click sobre la opción seleccionada. El evento click emula la acción de seleccionar con un evento táctil una opción del listado y cualquier botón de la pantalla o del terminal. Los métodos *onClick()* (para los botones) *onItemClick()* (para las filas del *listview*) reconocen el evento y ejecutan la funcionalidad correspondiente.

Por último están los métodos *onDestroy()* y *onBackPressed()* que se lanzan al cerrar la aplicación o pulsar el botón atrás (que en este caso también cierra la app). En ellos hay llamadas para interrumpir la reproducción de audio o el reconocimiento de voz que se puedan estar ejecutando y deberán ser interrumpidas.

3.3.4 Paquetes *adapters* y *definitions*

- En el paquete *adapters* se encuentra el adaptador que se asigna al listview que muestra las opciones en la *MainActivity*, la clase se llama *OptionsAdapter* y hereda de *BaseAdapter* de Android. Es por ello que implementa los métodos de dicha clase para gestionar la visualización de lista y los eventos que la afectan, como ir añadiendo las opciones según las recibe del servidor e ir actualizando las rows con el nuevo contenido. Veamos el UML que la define:

OptionsAdapter
Atributos

`private final String TAG="ADAPTER";`
`private Context c;`
`private ArrayList<String>l;`
`private LayoutInflater inflater;`

Métodos

`public OptionsAdapter(Context cx)`
`public int getCount()`
`public String getItem(int position)`
`public long getItemId(int position)`
`public View getView(int position, View convertView, ViewGroup parent)`

Tabla 36. UML clase OptionsAdapter.java

Observamos que el adaptador necesita un *Context*, asignar un *LayoutInflater* donde se crearán las filas de la lista y un *ArrayList* con los *String* a mostrar. En el constructor se inicializarán estas tres variables y el resto de métodos, que son *Override* de los de la clase madre, *BaseAdapter*, serán llamados dinámicamente por la propia

plataforma para crear la vista del listado. No será necesario pasar al constructor el listado de *String*, ya que puede obtenerlos de manera óptima de nuestros datos persistentes almacenados en la clase App, como ya comentamos.

Es importante decir que en esta clase, hay una clase estática llamada *ViewHolder*(que tendrá tantos atributos como *views* tenga una row, en nuestro caso sólo un*Textview)*, cuyo objeto es destinado a retener los datos en los objetos View, básicamente es como cachear los datos que ya se han creado y asociado a sus respectivas *Views* para mejorar el renderizado cuando se dibuje el *ListView*. Esto ayudará enormemente al funcionamiento del método que lleva todo el peso del *Adapter*, el método *getView().* Una vez que se ha determinado cuantos elementos hay disponibles, el *ListView* empieza a solicitar datos; pero sólo mostrará tantas filas como quepan en la pantalla y cuando el *ListView* pida más datos, el *ViewHolde*r empezará a reciclar los *Views* lo cual aumenta el rendimiento de la aplicación. Esto se hace de la siguiente forma: si el objeto *ViewconvertView*que recibe *getView()*es un valor no nulo significa que el *ListView* está reutilizando el objeto *View*, evitando inflar de nuevo el layout XML y hacer las llamadas a findViewById() para cada View de la fila (el TextView con una opción disponible). Al vincular el objeto *ViewHolder* a un *View* (con el método *setTag()*) se reciclará mucho más rápido la próxima vez que sea necesario mostrarla. Así, sólo hace falta recuperar el *ViewHolder* y asignarle los datos correspondientes al *View*, obtenidos, como decíamos de lo almacenado en nuestra clase *application* (*App.java*). Veamos el contenido del método principal:

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    ViewHolder holder;
    if (App.getInstance().getOptions() == null|| App.getInstance().getOptions().size() == 0) {
        Log.e(TAG, "Listado de opciones Vacío");
    }
    if (convertView == null) {
        convertView = inflater.inflate(R.layout.item_list, null);
        holder = new ViewHolder();
        holder.t = (TextView) convertView.findViewById(R.id.opcion);
        convertView.setTag(holder);
    } else
        holder = (ViewHolder) convertView.getTag();
    // Doy los valores al listview
    if(App.getInstance().getOptions().get(position).contains("@")){
        holder.t.setText(c.getResources().getString(R.string.contact)+App.getInstance().getOptions().get(position));
    }else{
        holder.t.setText(App.getInstance().getOptions().get(position));
    }
    return convertView;
}
```

Figura 39. Crear listado opciones

Lo único que faltará será desde la clase principal llamar al método *notifyDataSetChanged()* cada vez que queramos actualizar el listado de opciones, esto será cuando haya un nuevo electrodoméstico reconocido, cuando estén llegando las opciones desde el servidor. El flujo sería: recibo una nueva opción desde el servidor, la guardo con mi clase *App* y notifico al *adapter* de la *Listview* que el *ArrayList* con los *String* de las opciones ha sido modificado y por tanto me repinte el listado.

- En cuanto al paquete *definitions,* tiene dos clases: *Constants* que da tiene las constantes, como las credenciales de autorización de la app o los nombres de los parámetros a pasar en la petición de información al servidor, etc.; y Queries, que tiene un objeto *enumeration*el cual tiene valores como *SIGN_IN* y se utilizan en los métodos asíncronos para luego recoger su respuesta en el *onComplete* de una activity. Una cosa muy interesante que permite el enum es que al definir las constantes de este modo obtenemos *"type safety".* Si decimos que un método de una clase recibe un enum el compilador comprobará en tiempo de compilación que cuando lo usamos le pasemos realmente un valor de ese *enum*, cosa que con constantes definidas como *int* o *String* el compilador sólo comprueba que pasemos un *int* o *String* y por tanto podremos pasar cualquier valor aunque no sea una de las constantes esperadas por el método. A continuación se muestra el contenido de ambas clases.

Constants
Atributos

`public static final int APP_ID = 9422;`
`public static final String AUTH_KEY = "ac8bCmpQrFDCnDh";`
`public static final String AUTH_SECRET = "NDB78MUgHKwm-vx";`
`public static final String STATUS_NEW = "New";`
`public static final String STATUS_IN_PROCESS = "In Process";`
`public static final String STATUS_DONE = "Done";`
`public static final String CLASS_NAME_APPLIANCES = "Appliances";`
`public static final String CLASS_NAME_OPTIONS="Options";`
`public static final String ACTIONS = "actions";`
`public static final String IMAGE_QR = "imageQR";`
`public static final String NAME = "name";`
`public static final String SERIAL_NUMBER = "serialNumber";`
`public static final String BRAND = "brand";`
`public static final String CONTACT_US = "contactUs";`
`public static final String DEVICE_ID = "deviceID";`
`public static final String OPTION_ID="optionID";`
`public static final String OPTION_ACTION="action";`

Tabla 37. UML clase Constants.java

Queries
Atributos
`public enum Queries`

Tabla 38. UML clase Queries.java

3.3.5 Paquete *helper*

Este paquete es el que contiene los métodos de ayuda auxiliares que serán utilizados por la aplicación, divididos en los tres componentes que tiene el proyecto: el reconocimiento de QR, la reproducción de un audio con origen en una cadena de texto y el reconocimiento de voz y las peticiones de datos al servidor. Cabe mencionar que los métodos de estas clases serán estáticos para poder acceder a ellos sin necesidad de crear objetos de las mismas y sólo contendrán un atributo TAG para mostrar en los logs empleados en la depuración.

- Primeramente hablaremos de la clase que contiene los métodos relacionados con el audio.

MethodsAudio
Atributos
`private final String TAG="MethodsAudio";`
Métodos

`public static void configureText2Speech(TextToSpeechtextToSpeech, AudioManager am)`
`public static void speakText (TextToSpeechts, String str, Context c)`
`public static Intent launchVoiceRecognize()`

Tabla 39. UML clase MethodsAudio.java

El método *configureText2Speech()* sirve para configurar el volumen de la voz, el tono, la velocidad, en definitiva cualquier cosa que para mejorar el sonido de la reproducción de las opciones. El parámetro *AudioManager* es el que le dice a la plataforma nuestras opciones de configuración. *SpeakText()* es el que contiene la llamada al método *speak* de *TextToSpeech*, y configura los parámetros que se le pasan, en nuestro caso de la siguiente manera:

```
ts.speak(str, TextToSpeech.QUEUE_FLUSH, params);
```

Figura 40. Asignar texto a reproducir

El valor de la constante *QUEUE_FLUSH* hace que el *speak* no encole los textos recibidos, sino que vaya creando una nueva cadena de texto por cada vez que es llamado. Y por último el método *launchVoiceRecognize()*que es el que configura el *intent* de reconocimiento de voz con Google Voice y que devuelve el *intent* que será ejecutado por el método *startActivityForResult()*de la clase principal.

```
Intent intentActionRecognizeSpeech = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
intentActionRecognizeSpeech.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,"es-MX");
```

Figura 41. Lanzar reconocimiento de voz

- En segundo lugar tenemos la clase *MethodsImag.*

MethodsImage
Atributos
`private final String TAG="MethodsImage";`
Métodos
`public static Intent launchImageRecognize()` `public static ArrayList<String>getResultQRrecognize(String result)`

Tabla 40. UML clase MethodsImage.java

De la misma manera que en MethodsAudio, esta clase también tiene un método para configurar el *intent* para el reconocimiento de imágenes, *launchImageRecognize()* y actuará de la misma manera que su homólogo en audio.

```
Intent intent = new Intent("com.domoscan.SCAN");
intent.putExtra("SCAN_MODE", "QR_CODE_MODE,PRODUCT_MODE");
```

Figura 42. Configuración del modo de escaneo

De aquí hay varias cosas muy importantes. En primer lugar, veamos que se hace la llamada al intent con el nombre del paquete de nuestro proyecto, lo cual se hace para indicar que el reconocimiento de QR no llame a ninguna aplicación externa sino que se ejecute dentro de nuestra propia aplicación. Para ello es necesario también configurar una de las clases de la librería zXing, dentro del paquete com.google.zxing.android llamada Intents.java y en la que se deberá definir la variable ACTION también con el nombre de nuestro paquete. Además en el manifest de nuestro proyecto al definir la actividad de reconocimiento CaptureActivity, de zXing, será necesario indicarle el paquete com.domoscan.SCAN. Así nuestro proyecto queda listo para ejecutar el reconocimiento sin necesidad de recursos externos y evitando también al usuario tener que realizar más acciones para finalmente obtener el mismo resultado. También se le pasan otros parámetros que indican el tipo de *intent*, de escaneo, y el tipo de código que será reconocido, para que la librería sepa qué algoritmo de descifrado utilizar.

Por otro lado tenemos el método *getResultQRrecognize()*que recibe como parámetro la cadena de texto reconocida en la QR y la separa utilizando la clase *StringTokenizer* con una coma como delimitador. Así, devuelve un ArrayList con los parámetros necesarios para realizar la consulta al servidor de datos.

- Por último la clase *MethodsServerData* es la que gestiona las peticiones y respuestas de la aplicación y el servidor. Vemos que tiene otro atributo adicional que como explicaremos después, se utiliza en el método *getDataFromURL*.

MethodsServerData
Atributos
`private final String TAG="MethodsServerData";` `private static ArrayList<String>listFromUrl;`
Métodos

`public static void registerApp(QBCallback qc)`
`public static Appliance getAppliance(String brand, int id)`
`public static Options getOptions(int id)`
`public static String parseField(String field, QBCustomObjectcustomObject)`
`public static ArrayList<String>getDataFromURL(String url)`

Tabla 41. UML clase MethodsServerData

El método *registerApp(),* como dijimos es llamado en la Splash para realizar el registro de *DomoScan* en el servidor de Quickblox. Fundamentalmente realiza dos acciones: manda las credenciales y realiza la petición de autorización para la aplicación, tras lo cual el servidor le devuelve un token de autorización.

```
QBSettings.getInstance().fastConfigInit(String.valueOf(Constants.APP_ID), Constants.AUTH_KEY, Constants.AUTH_SECRET);
QBAuth.authorizeApp(qc, Queries.SIGN_IN);
```

Figura 43. Autorización en Quickblox

Los métodos *getAppliance()* y *getOptions()* funcionan de la misma manera. Veamos un fragmento de código del primero para explicar el funcionamiento:

```
QBCustomObjects.getObjects(Constants.CLASS_NAME_APPLIANCES, requestBuilder, new QBCallbackImpl() {
    @Override
    public void onComplete(Result result) {
        if (result.isSuccess()) {
            QBCustomObjectLimitedResult coresult = (QBCustomObjectLimitedResult) result;
            ArrayList<QBCustomObject> co = coresult.getCustomObjects();
```

Figura 44. Obtener datos desde el servidor

La llamada *getObjects()*invoca al módulo de *CustomObjects* del servidor Quickblox, pasándole en la consulta, el nombre del tipo de objeto que solicitamos, en este caso *Appliances* y el parámetro *requestBuilder.* Para construir este parámetro lo hacemos de la siguiente manera:

```
QBCustomObjectRequestBuilder requestBuilder = new QBCustomObjectRequestBuilder();
requestBuilder.eq(Constants.BRAND, qrResult.get(0).trim());
requestBuilder.eq(Constants.DEVICE_ID, qrResult.get(1).trim());
```

Figura 45. Configurar parámetros para la petición

Puesto que *qrResult* es el ArrayList resultado del método *getResultQRrecognize()* de la clase MethodsImage, contendrá los parámetros reconocidos de la QR, es decir la marca y el identificador. A partir de ellos construimos el *requestBuilder* que al pasárselo al método *getObjects()*lo que está diciendo es: "dame las filas de la BBDD cuya marca sea 'x' y con identificador 'y'", es decir envía todo lo necesario al servidor para que este realice la consulta a la base de datos. Como podemos ver con el parámetro de callback en el método, es una ejecución asíncrona que al terminar llama a su método *onComplete()* con el resultado obtenido, dentro del cual obtendremos un *ArrayList* de *QBCustomObject* que sólo deberá contener un elemento el cual transformaremos en un objeto *Appliance*, que es el objeto devuelto por el método *getAppliance().*Para convertir la respuesta en un objeto manejable por nuestra aplicación está el método *parseField()*, al cual sólo le tenemos que pasar el objeto devuelto por el servidor (*QBCustomObject*) y el nombre del campo que queremos obtener, por ejemplo el nombre del electrodoméstico, y así devolverá el *String* que podremos ir asignando a los diferentes atributos de nuestra clase modelo *Appliance*.

Por último tenemos el método *getDataFromURL(),* el cual tiene el propósito de generalizar el uso de la aplicación. En nuestro caso, tenemos el servidor de datos de Quickblox, donde subir la información y que sea disponible por la aplicación a través de QR con el formato que ya hemos comentado: <marca,id> . Pero planteemos la situación en la que un fabricante quiera tener su propio servidor de datos, ¿cómo se realizaría la petición puesto que la aplicación no tiene los datos de todos los servidores? Con este fin se crea este método, para que teniendo etiquetas QR que contengan una URL con la petición de información, tan sólo se tenga que lanzar una petición POST. Para ello utilizamos la librería android-async-http-1.4.4 con la que

podemos realizar la petición asíncrona sin necesidad de crear nuevos hilos, puesto que con crear un objeto de la clase *AsyncHttpClient* de la librería ya tendremos todo el hilo construido y listo para ejecutar. Veamos un fragmento del código:

```
AsyncHttpClient client = new AsyncHttpClient();
RequestParams rp = new RequestParams();

client.post(url,rp, new JsonHttpResponseHandler(){
 @Override
 public void onSuccess(JSONObject jObject){
     try {
        String name  = jObject.getString("name");
        listFromUrl.add(name);
        String contact  = jObject.getString("contact");
        listFromUrl.add(contact);
        String options  = jObject.getString("options");
        StringTokenizer qr=new StringTokenizer(options,",");
        while (qr.hasMoreTokens()){
            listFromUrl.add(qr.nextToken());
        }

    } catch (JSONException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
 }
 @Override
 public void onFailure(Throwable arg0){

 }
});
```

Figura 46. Petición a servidor externo

A simple vista se aprecia cómo se realiza el POST, sin necesidad de pasar parámetros adicionales puesto que ya van preconfigurados y los de la propia petición van en la url; y finalmente obtenemos un JSON que podremos tratar para obtener el nombre del electrodoméstico, el email de contacto técnico y las posibles acciones a realizar. Así, cualquier fabricante podrá montar su propio servidor de datos, manteniendo la estructura de los mismos y devolviendo un JSON con el formato especificado, crear una QR, que a diferencia del las QR elegidas en el proyecto con formato <marca,id> serán de tipo 'http://www.server.com/marca=x&id=y'. De esta manera se trató de hacer más funcional la aplicación facilitando la creación de datos y QR para ponerlos al servicio del usuario desde cualquier origen, con tan sólo mantener un formato, aunque es importante señalar que en este caso habría que tener en cuenta que al hacer una QR con una URL codificada, la cadena de caracteres es más larga que la tipo <marca,id>, por lo que habría que subir la versión de QR, lo cual podría perjudicar a la distancia de reconocimiento, punto clave en la realización de este proyecto. No obstante existen codificadores de urls que transforman una url muy larga en una mucho más corta, tal y como hace Twitter con sus extensas url.

3.3.6 Paquete *models*

Models es el paquete que contiene los modelos de datos (recordemos el elemento Modelo del patrón MVC descrito en capítulos anteriores). Las clases modelos son *Appliance* y *Options*. Decir de estas clases que tienen los atributos relacionados con las entradas en la BBDD y sus métodos de acceso *set()* y *get()*. *Appliance* serán los objetos que dan toda la información de un electrodoméstico y *Options* los que relacionan los identificadores numéricos de las acciones con la propia acción.

Appliance
Atributos

`private String name;`
`private int serialNumber;`
`private String brand;`
`private String contactUs;`
`private int devideID;`
`private ArrayList<Integer>actions;`
`private Date CreatedAt;`

Métodos

`public String getName()`
`public void setName(String name)`
`public int getSerialNumber()`
`public void setSerialNumber(intserialNumber)`
`public String getBrand()`
`public void setBrand(String brand)`
`public String getContactUs()`
`public void setContactUs(String contactUs)`
`public int getDevideID()`
`public void setDevideID(intdevideID)`
`public ArrayList<Integer>getActions()`
`public void setActions(ArrayList<Integer> actions)`
`public Date getCreatedAt()`
`public void setCreatedAt(Date createdAt)`

Tabla 42. UML clase Appliance.java

Options
Atributos

`private int optionID;`
`private String action;`
`private Date CreateAt;`

Métodos

`public int getOptionID()`
`public void setOptionID(intoptionID)`
`public String getAction()`
`public void setAction(String action)`
`public Date getCreateAt()`
`public void setCreateAt(Date createAt)`

Tabla 43. UML clase Options.java

3.3.7Fichero *AndroidManifest.xml*

Cualquier aplicación que deba correr sobre la plataforma Android, debe configurar el archivo *AndroidManifest.xml* en su directorio raíz. El manifiesto aporta información esencial para la ejecución de la aplicación en el terminal móvil. Entre otras funcionalidades están: dar permisos a la aplicación para los accesos a periféricos o a internet, declarar el nivel de API mínimo que requiere la aplicación, describir los

componentes que tiene, como las *activities*, etc. También aporta información de la versión de la aplicación, dato muy importante a la hora de subirla al Google Play, ya que si el *market* detecta que ya hay una versión subida y no hemos aumentado el número en la nueva *release*, no dejará subirla.

La etiqueta *<uses-sdk>* establece las versiones para las que se puede utilizar la aplicación. Debido a que para las pruebas de este proyecto se ha utilizado un terminal con *Gingerbread* (Android v2.3), el API mínimo es 9:

```
<uses-sdk
    android:minSdkVersion="9"
    android:targetSdkVersion="19" />
```

Figura 47. Configuración versión mínima de API

Como ya se ha ido viendo a lo largo del presente documento, la aplicación necesita tener acceso a internet, tanto por los temas de servidor como por los de audio, y poder controlar la cámara; por lo tanto mostramos a continuación cómo incluir estos permisos en el manifiesto.

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.FLASHLIGHT" />
<uses-permission android:name="android.permission.INTERNET" />
```

Figura 48. Asignación de permisos

Debido al uso de una clase tipo *application (App.java),* tal y como comentamos al explicar el funcionamiento de dicha clase, es necesario declararla ahora para que la aplicación la reconozca, lo cual se hace de la siguiente forma:

```
<application
    android:name="com.domoscan.app.App"
    android:allowBackup="true"
    android:icon="@drawable/icon"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
```

Figura 49. Configuración de clase App en el Manifest.xml

En cuanto a la declaración de los componentes de la aplicación, podemos ver que se declaran las activities, por ejemplo mostramos la definición de la pantalla principal y cómo el manifiesto la indica como primera en ejecutarse:

```
<activity
    android:name="com.domoscan.activities.SplashActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Figura 50. Declaración de Activity en el Manifest.xml

3.3.8 Otros aspectos: Idioma

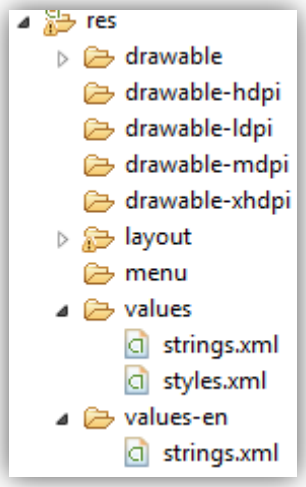


Figura 51. Carpeta idiomas

El tema del idioma en una aplicación puede decidir cómo de utilizada sea. Por ello aprovechando la opción que nos da Android de incluir varios idiomas, es bueno diseñar la aplicación para que pueda tener esta versatilidad. Si observamos la imagen a la derecha, vemos una carpeta llamada res que a su vez contiene varias llamadas values o values-en. En ellas podemos ver un archivo llamado strings.xml, el cual contiene los valores de los String mostrados en las pantallas del proyecto. En el caso de la que está contenida en la carpeta values, que será la de por defecto, contendrá las cadenas en castellano y la de values-en, en inglés. Veamos cómo sería:

```
<string name="app_name">DomoScan</string>
<string name="device">ELECTRODOMÉSTICO</string>
<string name="actions">ACCIONES</string>
<string name="status">Activar Escaneo QR</string>
```

```
<string name="app_name">DomoScan</string>
<string name="device">APPLIANCES</string>
<string name="actions">ACTIONS</string>
<string name="status">Enable QR Scan</string>
```

Figura 52. Declaración de String en varios idiomas

De esta manera, si el teléfono está configurado en inglés, la propia plataforma Android seleccionará los textos incluidos en la carpeta values-en.

3.4 Evaluación y Resultados

Finalmente se ha llevado a cabo una fase de verificación de resultados a través de la realización de pruebas; aunque realmente también se han ido reproduciendo durante todo el proceso de implementación, cada vez que se cumplimentaba un hito marcado, para corregir los errores detectados y no llegar al final con un sistema completo sin probar. Además, cabe mencionar que en el planteamiento del problema también se llevaron a cabo pruebas para la decisión del tipo de etiqueta a utilizar, siguiendo las directrices de eficacia en el reconocimiento y las distancias entre etiqueta y dispositivo.

Una vez desarrollada la aplicación, hemos de comprobar que funciona correctamente, para lo cual es necesario llevar a cabo una batería de pruebas quecomprueben si se cumplen todos los requisitos funcionales descritos en el apartado de de análisis. Se presentan a continuación los test funcionales realizados, en tablas autoexplicativas, para intentar hacer más dinámica la lectura.

PRUEBA FUNCIONAL 01			PF_01
Requisito que satisface	SR_F_01, SR_F_03, SR_F_06	Resultado	Satisfactorio
Descripción	Instalación de *DomoScan* en el terminal Samsung Galaxy S.		
Procedimiento	1. Conexión PC-Smartphone mediante cable USB. 2. Se lanza la aplicación desde el IDE Eclipse a través de adb server con conectividad en modo desarrollador. 3. Se prueba también a instalar la aplicación copiando y pegando el apk en el terminal conectado en modo almacenamiento masivo.		

Tabla 44. Prueba funcional PF_01

PRUEBA FUNCIONAL 02			PF_02
Requisito que satisface	SR_F_02, SR_NF_12	Resultado	Satisfactorio
Descripción	Abrir la aplicación frente a un código QR.		
Procedimiento	1. Se pone QR de 8.4x8.4 cm sobre un frigorífico. 2. Se lanza la aplicación. 3. Se coloca el Smartphone a una distancia de 2 metros y se va acercando hasta reconocer QR.		

Tabla 45. Prueba funcional PF_02

Para la PF_02 es necesario un poco más de explicación, ya que el tema de las distancias de reconocimiento ha sido un punto muy trabajado en el proyecto puesto que era un requisito fundamental debido a la distancia que establece una silla de ruedas con una etiqueta sobre un electrodoméstico. Las pruebas realizadas en este aspecto siempre parten de una distancia más grande que la calculada teóricamente con la fórmula que comentamos en el apartado de QR al principio de esta memoria, para una QR de 8.5cm aprox. Se hacen así porque a través de la realización de múltiples pruebas de reconocimiento, se verificó que la distancia calculada siempre era inferior a la que se podía obtener en realidad con unas buenas condiciones de contraste e iluminación. Por ello, la distancia de reconocimiento que se ha llegado a conseguir es de 1.9m, que es aceptable para los requisitos del proyecto. Se han obtenido mayores distancias (de hasta 2.23m); pero para un tiempo de espera de entorno a 30-40 segundos, cosa que se piensa, perjudicaría la experiencia de usuario. No obstante, es necesario señalar que el terminal del que se disponía para las pruebas es ya antiguo (Samsung Galaxy S) y es de esperar que el comportamiento de DomoScan en otros Smartphone con cámaras de mayor resolución y flash sea aún mejor, ya que se pudieron realizar varias pruebas ocasionales en un Samsung Galaxy SIII y se obtuvieron distancias de hasta 2.3m con reconocimiento prácticamente instantáneo.

PRUEBA FUNCIONAL 03			PF_03
Requisito que satisface	SR_F_04, SR_F_05, SR_F_07, SR_F_09, SR_NF_14	Resultado	Satisfactorio
Descripción	Obtención de datos desde el servidor Quickblox.		
Procedimiento	1. Se lanza la aplicación y se reconoce una QR. 2. Se observa el listado de opciones incrementar conforme se van recibiendo los datos del servidor. 3. Se comprueba la aparición del botón de contacto técnico y el nombre general del electrodoméstico reconocido.		

Tabla 46. Prueba funcional PF_03

PRUEBA FUNCIONAL 04			PF_04
Requisito que satisface	SR_F_08	Resultado	Satisfactorio
Descripción	Comprobación de reproducción de audio con las opciones correspondientes.		
Procedimiento	1. Se abre la aplicación, reconoce QR y muestra las opciones. 2. Tras concluir, se espera a la escucha del audio para verificar que las opciones 'leídas' por la aplicación son las correctas.		

Tabla 47. Prueba funcional PF_04

PRUEBA FUNCIONAL 05			PF_05
Requisito que satisface	SR_F_09	Resultado	Satisfactorio
Descripción	Pulsando el botón de soporte.		
Procedimiento	1. Se acciona el botón soporte, a través de una orden de voz. 2. Se verifica que se abre la aplicación de redacción de emails en nuestro Smartphone y se puede enviar un email a la dirección facilitada desde el servidor.		

Tabla 48. Prueba funcional PF_05

PRUEBA FUNCIONAL 06			PF_06
Requisito que satisface	SR_F_10, SR_NF_13	Resultado	Satisfactorio
Descripción	Verificación del reconocimiento de voz.		
Procedimiento	1. Tras la escucha de las opciones se da una orden por voz. 2. El usuario de prueba está sentado y tiene el móvil a una distancia de hasta medio metro.		

Tabla 49. Prueba funcional PF_06

PRUEBA FUNCIONAL 07			PF_07
Requisito que satisface	SR_F_10, SR_F_11	Resultado	Satisfactorio
Descripción	Manejo de la aplicación tanto por voz como por taps.		
Procedimiento	1. Se abre la aplicación, reconoce QR, audio con opciones. 2. Ejecución de órdenes por voz o también con un tap. 3. Visualización de mensajes de reconocimiento, apertura de la aplicación email, ejecución de un nuevo escaneo, cierre de la aplicación, y cualquier acción disponible.		

Tabla 50. Prueba funcional PF_07

De igual forma que se construyó la matriz de trazabilidad que relacionaba los requisitos de software con los de usuario, ahora podemos hacer lo mismo con las pruebas funcionales y los requisitos de software, de manera que puesto que los requisitos de software aseguraban el cumplimiento de los requisitos de usuarios, al verificar ahora los de software con las pruebas funcionales, el funcionamiento completo se verá ratificado. Por ello elaboramos ahora la matriz de trazabilidad de funcionalidades.

	PF_01	PF_02	PF_03	PF_04	PF_05	PF_06	PF_07
SR_F_01	X						
SR_F_02		X					
SR_F_03	X						
SR_F_04			X				
SR_F_05			X				
SR_F_06	X						
SR_F_07			X				
SR_F_08				X			
SR_F_09			X		X		
SR_F_10						X	X
SR_F_11							X
SR_NF_12		X					
SR_NF_13						X	
SR_NF_14			X				

Tabla 51. Matriz de trazabilidad: Requisitos de software – Pruebas funcionales

Se puede comprobar que todas las filas contienen al menos un aspa, por lo que la funcionalidad recogida en los requisitos queda comprobada con el resumen de tests expuestos anteriormente. Es muy importante añadir, que las matrices de trazabilidad como las expuestas en este documento, serán herramienta fundamental a la hora de realizar cambios en un proyecto, ya que con un simple vistazo se podrá conocer a qué otros requisitos afecta, viendo qué pruebas fallarán y así saber el coste que pueden suponer las modificaciones pedidas.

Capítulo 4. Planificación y Presupuesto

En este capítulo se expone la planificación llevada a cabo para el desarrollo de las distintas fases del proyecto. También se detallará un presupuesto que de una idea del coste que tendría la implementación del resultado final.

4.1 Planificación

La planificación de un proyecto es fundamental para el buen desarrollo de éste. Cumplir con los plazos previstos, saber qué partes son críticas si no se terminan a tiempo y que relación guardan con el resto de actividades, garantizará el resultado exitoso del mismo.

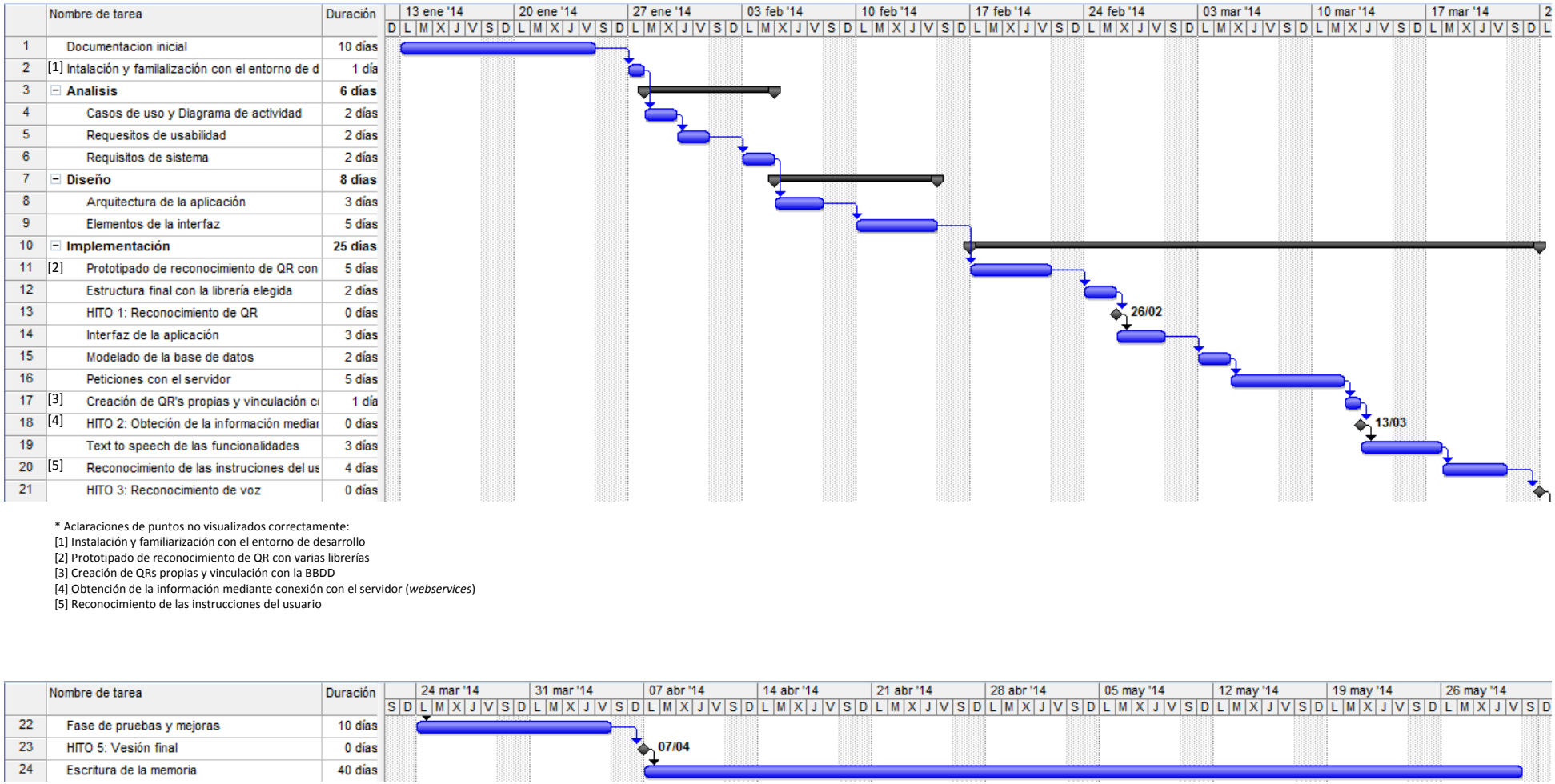
Para seguir la planificación es necesario que los puntos estén definidos y que el desarrollador pueda consultarla de un vistazo y tener una idea concisa y clara. Para ello, se utiliza el diagrama de *Gantt*, el cual sitúa de forma secuencial las diferentes etapas del proyecto a lo largo de una línea temporal que representa el tiempo total para la consecución del proyecto.

Primeramente veamos una tabla que resuma los hitos principales del proyecto, sus fechas de comienzo y fin y la duración de los mismos, para en la siguiente página, visualizar la planificación completa en el *Gantt.*

ACTIVIDAD	FECHA INICIO	FECHA FIN	DURACIÓN (DÍAS)
Estudio inicial	13/01/14	27/01/14	11
Análisis	28/01/14	04/02/14	6
Diseño	05/02/14	14/02/14	8
Implementación	17/02/14	21/02/14	25
Pruebas y correcciones	24/03/14	04/04/14	10
Elaboración de la memoria	07/04/14	30/05/14	40

Tabla 52. Planificación de las tareas globales

4.1.1 Diagrama de Gantt



* Aclaraciones de puntos no visualizados correctamente:
[1] Instalación y familiarización con el entorno de desarrollo
[2] Prototipado de reconocimiento de QR con varias librerías
[3] Creación de QRs propias y vinculación con la BBDD
[4] Obtención de la información mediante conexión con el servidor (*webservices*)
[5] Reconocimiento de las instrucciones del usuario

Figura 53. Diagrama de Gantt

4.2 Presupuesto

Para realizar el presupuesto del proyecto, lo dividiremos en dos tipos de costes: el material y el trabajo humano. De acuerdo a la planificación expuesta anteriormente podemos obtener las horas empleadas y hacer una estimación monetaria de costes. Vamos a desglosarlo en los siguientes apartados.

4.2.1 Costes materiales

El material informático utilizado en la realización de este proyecto ha sido:

CONCEPTO	CARACTERÍSTICAS
Ordenador de mesa	Fabricante: Packard Bell Procesador: Intel Core Quad Q6600 2.40 GHz Memoria RAM: 4,00 GB Sistema Operativo: Windows 7
Teléfono móvil	Fabricante: Samsung Modelo: Galaxy Mini Procesador: ARM v6 600 MHz Memoria RAM: 384 MB S.O. : Android 2.3 (Gingerbread)

Tabla 53. Materiales empleados

En la siguiente tabla se detalla el coste de equipamiento teniendo en cuenta el precio, el tiempo de uso y el periodo de amortización de los mismos.

CONCEPTO	PRECIO	Tº DE USO	Tº AMORTIZACIÓN	AMORTIZACIÓN
Ordenador de mesa	1000,00 €	4,5 Meses	60 Meses	75 €
Teléfono móvil	300,00 €	3 Meses	12 Meses	75 €
Total	**1300,00 €**			**150 €**

Tabla 54. Costes materiales

4.2.2 Coste de personal

La duración del desarrollo completo de la aplicación *DomoScan* asciende a un total de 60 días por 8 h/día de trabajo, suponiendo un total de 480 horas repartidas en 3 figuras diferenciadas: un analista, un diseñador y un programador. Aunque muchas veces en la realidad, así como en este proyecto, los 3 papeles los desempeña la misma persona, para hacer una correcta estimación del coste distinguiremos las 3 categorías:

CATEGORÍA PROFESIONAL	RETRIBUCIÓN (€/hora)	HORAS TRABAJADAS (h/persona)	TOTAL
Analista	65 €	136 h	8.840 €
Diseñador	55 €	64 h	3.520 €
Programador	45 €	280 h	12.600 €
TOTAL	-	480 h	**24.960 €**

Tabla 55. Costes de personal

El precio por horas está basado en el coste real que he podido conocer en la empresa para la que trabajo.

4.2.3 Coste total

Teniendo en cuenta los costes establecidos en los desgloses anteriores, se llega al precio final de la aplicación.

CONCEPTO	PRECIO
Costes materiales	150 €
Costes de personal	24.960 €
Subtotal	25.110 €
I.V.A (21%)	5.273,1€
Total	**30.383,1€**

Tabla 56. Coste total

→ Como muestra la tabla, el presupuesto total para tener funcionando la aplicación *DomoScan* es de **30.383,1 €**.

Capítulo 5. Conclusiones y Línea de futuro

En las próximas páginas que constituyen el apartado final del documento, se comentarán las conclusiones obtenidas de la realización del proyecto; así como se expondrán algunas ideas de posibles funcionalidades adicionales para proyectar una línea de futuro que pueda resultar interesante.

5.1 Conclusiones

Una vez concluido el desarrollo del proyecto, es momento de hacer balance y ver si los objetivos planteados han sido logrados. La finalidad que buscaba el proyecto era realizar una aplicación Android que reconociera etiquetas situadas en electrodomésticos y se pudiera manejar a través de órdenes por voz, para poder ser utilizada por personas con una discapacidad física en la movilidad; funcionamiento que ha quedado probado con las pruebas realizadas.

Dicho funcionamiento y su fase de test confirman directamente la consecución de otros objetivos secundarios como:

- ✓ Se ha almacenado y recuperado la información de una BBDD remota.
- ✓ Una vez obtenidos los datos, se crea y reproduce una cadena de audio con las opciones disponibles.
- ✓ Tras ello, se activa el reconocimiento de voz para captar la opción que desea el usuario, y se ejecuta la acción correspondiente.
- ✓ En la etapa de evaluación de resultados se han llevado a cabo repetidas veces las pruebas que se plantearon hacer en la fase inicial, con un gran empeño de las mismas para constatar el funcionamiento de la aplicación.
- ✓ La aplicación es por tanto funcional tanto en el modo tradicional mediante taps como sin necesidad de tocarla, a través de las órdenes por voz; y en cualquier caso con una navegabilidad muy sencilla.

Por otro lado, los test que se fueron haciendo a la par del desarrollo para ir probando cada parte implementada, llevaron a la conclusión de no hacer un servicio que corriera siempre bajo la aplicación para el reconocimiento de voz, puesto que como veíamos en los requisitos, las órdenes por voz no debían interceder negativamente en el funcionamiento de la app. Además, de una fase inicial de ejecución de demos para emular un funcionamiento final, surgió uno de los objetivos del proyecto, la reproducción de la cadena de audio para facilitar el entendimiento por parte del usuario de las opciones disponibles sin necesidad de leer la pantalla. Por todo ello queda demostrado que tras el análisis y pruebas parciales durante la realización del trabajo, el testeo ha ido modificando positivamente el código para adaptarlo a las necesidades descubiertas a fin de favorecer la experiencia de usuario.

El objetivo quizá más importante y complicado de lograr era el tema de la distancia etiqueta-Smartphone. Era fundamental tener en cuenta la limitación que

suponía tener un terminal móvil instalado en una silla de ruedas sin posibilidad de acercarlo a la etiqueta para un correcto reconocimiento. Por ello se jugó con los factores descritos como la iluminación, el nivel de corrección de errores, la cantidad de información, etc. para conseguir un resultado como el obtenido, que como se vio en el capítulo de pruebas funcionales, da unas distancias de reconocimiento de hasta 2.2m aproximadamente para un QR de 8.5cm de lado, lo cual supera ampliamente la distancia objetivo de 1.5m.

Por último, la utilización del MVC ha sido muy importante para los 2 objetivos que quedan por nombrar:

- ✓ Hacer una estructura de proyecto enfocada a que pudiera ser la base de una línea futura más compleja.
- ✓ Desarrollar el código de la aplicación de una manera clara, eficiente y estructurada.

La estructuración del proyecto con el modelo y facilitado por la forma de desarrollar en la plataforma Android, al tener la lógica separada de la interfaz, ha favorecido la consecución de estos objetivos, ya que en cualquier momento se pueden añadir pantallas con tan sólo añadir activities y xml's que las implementen, pudiendo utilizar fácilmente la lógica desarrollada en otras clases independientes ya creadas. Además de cara a una línea futura más compleja que deba integrarse con servidores de datos ajenos al administrador de la aplicación, la inclusión de peticiones post genéricas hace que el proyecto esté abierto a continuarse para mejorarlo e integrarlo con el sistema completo.

Personalmente, existían 2 objetivos muy importantes para mí:

- ✓ desarrollar una aplicación que fuera capaz de superar los impedimentos marcados por la discapacidad de los usuarios
- ✓ hacerlo poniendo todo el empeño en mejorar mi forma de programar, en muchas ocasiones desorganizada y desaprovechando los recursos a mi alcance

La primera era una cuestión personal, puesto que desde que empecé la carrera siempre quise aprender la manera de poner la tecnología al servicio de las personas pensando en las posibilidades que se podrían abrir mejorando su calidad de vida.

La segunda es un tema profesional, puesto que me encanta programar y quería dedicarme a ello. Para conseguirlo, tenía que mejorar utilizando buenas prácticas y siendo más organizada con el código, por lo que el proyecto me ha servido enormemente para trabajar este aspecto, lo cual seguiré haciendo.

5.2 Línea de futuro

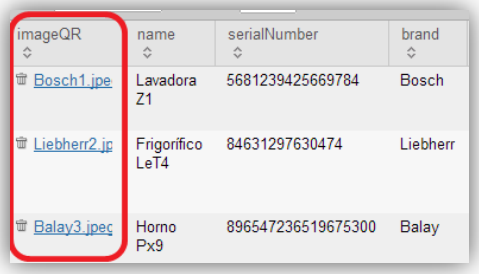
Este capítulo está dedicado a comentar las posibilidades que existen para completar y mejorar la aplicación *DomoScan.* Pasemos a comentarlas:

- INTEGRACIÓN CON EL SISTEMA COMPLETO

Para el funcionamiento total de la idea planteada, habría que desarrollar un software que se instalase en los electrodomésticos y realizar una integración con la aplicación del Smartphone para mandar, reconocer y ejecutar las órdenes y que los aparatos hicieran lo que el usuario desee. Dichas órdenes se mandarán desde el móvil mediante Wi-Fi, o podría también hacerse por Bluetooth. Así, se conseguiría un sistema de domótica asistencial completo, que proveerá a los usuarios de una pequeña facilidad a la hora de desempeñar algunas tareas cotidianas.

- DESCARGA DE ETIQUETAS

Esta posibilidad es factible gracias a que en la parte del servidor, como vemos en la figura de la derecha, la imagen de la QR está guardada en formato jpeg y asociada a sus características. Por ello, se creará una *actionbar* o un botón en la aplicación que ofrezca la opción al usuario de descargar la imagen a su terminal, para poder imprimirla él mismo y sustituirla por la que tuviera, si ésta estuviera dañada o se hubiera extraviado, sin necesidad de contactar con el servicio técnico.



imageQR	name	serialNumber	brand
Bosch1.jpe	Lavadora Z1	5681239425669784	Bosch
Liebherr2.jp	Frigorífico LeT4	84631297630474	Liebherr
Balay3.jpeg	Horno Px9	896547236519675300	Balay

Figura 54. Entradas en la BBDD

- MODO OFFLINE

Para agilizar más aún el funcionamiento de *DomoScan* se podría implementar una BBDD en local con un SQLite para que una vez reconocida la QR, la aplicación reconozca que ya tiene la información de ese aparato y no necesita contactar con el servidor, sino que ya tiene todos los datos y directamente se activara la escucha de las opciones y el reconocimiento de voz. De esta manera, aunque la petición post y la obtención de la respuesta es bastante rápida, se podría conseguir mejores resultados, ya si el móvil en ese momento no tiene conexión a internet, o falla, el funcionamiento sería correcto igualmente.

- MIGRACION A OTRAS PLATAFORMAS

Como ya vimos en el estudio de la plataforma Android, hoy en día es la más utilizada por los usuarios de Smartphone; pero para ofrecer esta aplicación a un mayor

número de usuarios, habría que desarrollarla para el resto de plataformas, en concreto para su, actualmente, mayor competidor que es iOS. De esta manera, *DomoScan* podría ser utilizada con independencia del móvil que tuviera el usuario.

- GOOGLE GLASS ("GLASS")

Figura 55. Google Glass

Las Google Glass son un dispositivo de visualización desarrollado por Google, que salió a la venta para desarrolladores en 2013 y para el consumidor final sólo disponibles el 15 de abril de 2014. Las gafas llevan cámara de 5MP de resolución, un touchpad en el costado que permite a los usuarios controlar el dispositivo mediante gestos como desplazar o tocar, tiene todas las características principales de *Smartphone* como giroscopio, acelerómetro, Bluetooth, conexión Wi-Fi, etc., posee también sensores de luz ambiente y de proximidad, un sistema de inducción ósea para la transmisión del sonido, entre otras muchas características. Una muy interesante relacionada con la funcionalidad del proyecto es el reconocimiento de órdenes sin necesidad de tocar nada, sólo mediante gestos como el touchpad y el reconocimiento de la voz. Para activar este modo, el usuario debe inclinar la cabeza hacia arriba en un ángulo previamente configurado y decir “Ok, Glass”. A partir de ese momento los usuarios pueden decir una acción como hacer una búsqueda en internet o enviar un mensaje y las gafas ejecutarán la opción mencionada. Google publicó varios videos demostrativos del funcionamiento de las Google Glass, por lo que si al lector le resulta interesante, este es uno de ellos: https://www.youtube.com/watch?v=v1uyQZNg2vE

Si pensamos en el mundo de posibilidades que abre vemos que el objetivo de este proyecto sería superado con creces y la experiencia de usuario mejoraría indescriptiblemente, al no tener que instalar un teléfono móvil en una silla de ruedas, sino que con unas ‘simples’ gafas el usuario podría controlar sus electrodomésticos sin la menor dificultad. Además dada la relación Google – Android, el S.0 de las Google Glass se basa en la versión 4.0.4 de la plataforma por lo que creo que no sería muy complejo adaptar la aplicación desarrollada para este proyecto para que funcionase en las gafas.

A pesar de que su precio es de 1500$ más impuestos y que de momento sólo están disponibles para desarrolladores residentes en EEUU, (aunque se cree que podrían llegar a España a lo largo del 2014 y disminuir su precio hasta 400€) las Google Glass, por todas las características descritas, son claramente la línea de futuro a seguir en este proyecto.

ANEXO I: Más información sobre el estado del arte

❖ Clasificación de dispositivos móviles

Como hemos visto en el apartado 2.1.1 de la memoria, dentro de esta definición se incluyen un gran y variado número de dispositivos y funcionalidades. Por ello en el año 2005, *T38* y *DuPont Global MobilityInnovationTeam* propusieron estándares para clasificar más concretamente los distintos dispositivos móviles. Dicha clasificación podemos verla a continuación.

Tenemos 3 estándar para clasificar los dispositivos móviles: (16) (17)

- Dispositivo Móvil de Datos Limitados
  (Limited Data Mobile Device)

Básicamente poseen una pantalla pequeña, generalmente de tipo texto y los servicios que ofrecen suelen ser únicamente SMS y acceso WAP. Los teléfonos móviles clásicos (inalámbricos) cuya principal funcionalidad es el acceso a la red de telefonía son un ejemplo de este subtipo.

Figura 56. Móvil de datos limitados

- Dispositivo Móvil de Datos Básicos
  (Basic Data Mobile Device)

En esta segundo tipo encontramos ya una pantalla de tamaño medio (entre 120 x 120 y 240 x 240 píxeles), con menú o navegación basada en iconos. Ofrecen servicios tales como acceso a email, listado de direcciones, SMS, y en algunos casos también un navegador web básico. Con esta definición el ejemplo claro son los teléfonos inteligentes, Smartphone; pero también las BlackBerry son incluidas en esta clasificación.

Figura 57. Móvil de datos básicos

- Dispositivo Móvil de Datos Mejorados
  (Enhanced Data Mobile Device)

Los dispositivos que engloba este estándar se caracterizan por tener pantallas de medianas a grandes (por encima de los 240 x 120 píxeles), navegación de tipo stylus (los típicos lápices en vez de utilizar el dedo sobre la pantalla), y que

Figura 58. Móvil de datos mejorados

ofrecen las mismas características que el "Dispositivo Móvil de Datos Básicos" además de aplicaciones nativas como aplicaciones de Microsoft Office Mobile (Word, Excel, PowerPoint) y aplicaciones corporativas usuales, en versión móvil, como Sap, portales intranet, etc. Este tipo de dispositivos incluyen los S.O. como Windows Mobile, un ejemplo son las Pocket PCs (también conocidas por PDAs).

❖ Sistemas operativos para Smartphone

- **iOS**

Con este sistema operativo, Apple ofrece un rendimiento excelente gracias a la integración hardware-software. Su principal baza es ofrecer una experiencia única al usuario mediante un funcionamiento sencillo. El apoyo de los desarrolladores está siendo fundamental para este S.O., ya que en los últimos años son muchos los que están apostando por hacer aplicaciones para el Apple Store.

Figura 59. iOS

  - Ventajas
    - Twitter es directamente integrado en el iPhone.
    - Los iMessages, comunicación exclusiva entra iPhones.
    - Siri, un programa de reconocimiento de voz.
    - Facetime, permite realizar videollamadas.
  - Inconvenientes
    - Funcionalidades compatibles únicamente entre iPhones, por ejemplo: Facetime o iMessages.
    - Una gama de aplicaciones y funcionalidades sólo accesibles mediante el pago.
    - Un punto débil de iOS es la absoluta dependencia de iTunes para sincronizar los datos, ya que no es posible conectarlo por USB y gestionarlo igual que cualquier memoria externa.

- **Symbian**

Nokia era la marca que dominaba el mercado de la telefonía en España y con Symbian en su día revolucionó este mercado. Sin embargo la compañía no supo adaptarse a los cambios en este ámbito y su sistema se ha quedado bastante lejos de la oferta de su competencia. Symbian forma parte de terminales básicos en comparación con los que han ido sacando sus competidores, a pesar de tener un servicio de navegación GPS gratuito y eficaz capaz de distinguir si vamos a pie o en coche.

Figura 60. Symbian

- Ventajas
  - Sistema operativo fiable (presencia desde hace 10 años en el mercado).
  - Mayor duración de la batería. Tiene más autonomía.
  - Un sistema multitarea bien desarrollado.
- Inconvenientes
  - El mercado de aplicaciones.
  - Interfaz poco estética.

En los últimos tiempos, parece que Nokia está tratando de evolucionar hacia Windows Phone como su renacer en el mercado de los Smartphone.

- **Windows Phone**

Es el S.O. que lanzó Microsoft para Smartphone, cuyas últimas versiones (Windows Phone 7 y 8) ofrecen una muy intuitiva y espectacular interfaz, además de que para usuarios de herramientas de Windows como Hotmail u Office es una muy buena opción ya que aporta una integración con ambas muy lograda. Además Windows 8 provee una perfecta fusión entre PCs y Smartphone.

Figura 61. Windows Phone

- Ventajas
  - Ejecución rápida.
  - Interfaz con el pack Office.
- Inconvenientes
  - Catálogo bastante escaso de aplicaciones.
  - Sólo disponible en terminales de gama alta.
  - Ausencia de multitarea.
  - Ausencia de la tecnología Flash.

- **BlackBerry**

Fue desarrollado por RIM para los dispositivos BlackBerry. Permite multitarea (varios procesos a la vez compartiendo o no procesador) y soporta diferentes métodos de entrada adoptados por RIM para su uso en computadoras de mano, como las pantallas táctiles o el touchpad.

Figura 62. BlackBerry

El SO BlackBerry está claramente orientado a su uso profesional como gestor de correo electrónico y agenda. La última versión, BlackBerry OS 10, permite a los usuarios poder instalar aplicaciones Android, aunque ya se ha anunciado que aún es pronto para asegurar que cualquiera funcionará.

- Ventajas
  - Una navegación web rápida.
  - Muy práctico para leer los emails.
  - Social Feeds (multiposteo en las redes sociales).
  - BlackBerry Messenger.
- Inconvenientes
  - Su principal desventaja es la ergonomía.

- **Firefox OS**

S.O. de Mozilla y apoyado por Telefónica, basado en HTML5 con kernel Linux y de código abierto. Está diseñado para permitir a las aplicaciones comunicarse directamente con el hardware del dispositivo usando principalmente JavaScript.

Figura 63. Firefox OS

- Ventajas
  - Sistema ligero basado en el navegador.
  - Consume pocos recursos por lo que el uso de la batería disminuye, frente a otros sistemas operativos.
  - Al ser sistema abierto puede ser modificado por usuarios, desarrolladores u operadoras (al estilo de Android).
  - Al estar basado en HTML5 que ya es un estándar web, se pueden ejecutar aplicaciones ya desarrolladas sin restricciones por parte del sistema operativo.
- Inconvenientes
  - Es completamente nuevo y tendremos que ver cómo funcionan realmente las integraciones de los fabricantes en los terminales
  - Aún no tiene suficientes desarrolladores para convertirse en un S.O. puntero.
  - Tradicionalmente el HTML5 no funciona bien sin conexión a internet por lo que el modo offline quizá pueda no ser demasiado óptimo.

❖ Arquitectura Android

A continuación explicamos cada capa desde su base hasta el nivel más alto.

- **Kernel Linux**

El núcleo actúa como una capa de abstracción entre el hardware y el resto de las capas de la arquitectura. El desarrollador no accede directamente a esta capa, sino que debe utilizar las librerías disponibles en capas superiores. Así se evita tener que conocer las especificaciones de cada teléfono; si se precisa usar la cámara es el propio S.O. el encargado de gestionar la que tenga su terminal. Para cada elemento de

hardware del teléfono existe un controlador (o driver) dentro del kernel que permite utilizarlo desde el software.

- **Librerías**

Es la capa que se sitúa por encima del kernel. Está compuesta por las librerías nativas de Android. Están escritas en C o C++ y compiladas para la arquitectura hardware específica del teléfono. Éstas normalmente están hechas por el fabricante, quien también se encarga de instalarlas en el dispositivo antes de ponerlo a la venta. El objetivo de las librerías es proporcionar funcionalidad a las aplicaciones para tareas que se repiten con frecuencia, evitando tener que codificarlas cada vez y garantizando que se llevan a cabo de la forma "más eficiente".

Entre las librerías incluidas habitualmente encontramos OpenGL (motor gráfico), Bibliotecas multimedia (formatos de audio, imagen y video), Webkit (navegador), SSL (cifrado de comunicaciones), FreeType (fuentes de texto), SQLite (base de datos), entre otras.

- **Entorno de ejecución**

Como podemos apreciar en el diagrama de la arquitectura, el entorno de ejecución de Android no se considera una capa en sí mismo, dado que también está formado por librerías, las cuales poseen las funcionalidades habituales de Java así como otras específicas de Android.

El componente principal del entorno de ejecución de Android es la máquina virtual Dalvik. Las aplicaciones se codifican en Java y son compiladas en un formato específico para que esta máquina virtual las ejecute. La ventaja de esto es que las aplicaciones se compilan una única vez y de esta forma estarán listas para distribuirse con la total garantía de que podrán ejecutarse en cualquier dispositivo Android que disponga de la versión mínima del sistema operativo que requiera la aplicación.

Cabe aclarar que Dalvik es una variación de la máquina virtual de Java, por lo que no es compatible con el bytecode de Java. Java se usa únicamente como lenguaje de programación, y los ejecutables que se generan con el SDK de Android tienen la extensión .dex que es específico para Dalvik, y por ello no podemos correr aplicaciones Java en Android ni viceversa.

- **Framework de aplicaciones**

La siguiente capa está formada por todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones. La mayoría de los componentes de esta capa son librerías Java que acceden a los recursos de las capas anteriores a través de la máquina virtual Dalvik. Siguiendo el diagrama encontramos:

1. Activity Manager. Se encarga de administrar la pila de actividades de nuestra aplicación así como su ciclo de vida.

2. Windows Manager. Se encarga de organizar lo que se mostrará en pantalla. Básicamente crea las superficies en la pantalla que posteriormente pasarán a ser ocupadas por las actividades.

3. Content Provider. Esta librería es muy interesante porque crea una capa que encapsula los datos que se compartirán entre aplicaciones para tener control sobre cómo se accede a la información.

4. Views. En Android, las vistas los elementos que nos ayudarán a construir las interfaces de usuario: botones, cuadros de texto, listas y hasta elementos más avanzados como un navegador web o un visor de Google Maps.

5. Notification Manager. Engloba los servicios para notificar al usuario cuando algo requiera su atención mostrando alertas en la barra de estado. Un dato importante es que esta biblioteca también permite jugar con sonidos, activar el vibrador o utilizar los LEDs del teléfono en caso de tenerlos.

6. Package Manager. Esta biblioteca permite obtener información sobre los paquetes instalados en el dispositivo Android, además de gestionar la instalación de nuevos paquetes. Con paquete nos referimos a la forma en que se distribuyen las aplicaciones Android, estos contienen el archivo .apk, que a su vez incluyen los archivos .dex con todos los recursos y archivos adicionales que necesite la aplicación, para facilitar su descarga e instalación.

7. Telephony Manager. Con esta librería podremos realizar llamadas o enviar y recibir SMS/MMS, aunque no permite reemplazar o eliminar la actividad que se muestra cuando una llamada está en curso.

8. Resource Manager. Con esta librería podremos gestionar todos los elementos que forman parte de la aplicación y que están fuera del código, es decir, cadenas de texto traducidas a diferentes idiomas, imágenes, sonidos o layouts. En un post relacionado a la estructura de un proyecto Android veremos esto más a fondo.

9. Location Manager. Permite determinar la posición geográfica del dispositivo Android mediante GPS o redes disponibles y trabajar con mapas.

10. Sensor Manager. Nos permite manipular los elementos de hardware del teléfono como el acelerómetro, giroscopio, sensor de luminosidad, sensor de campo magnético, brújula, sensor de presión, sensor de proximidad, sensor de temperatura, etc.

11. Cámara: Con esta librería podemos hacer uso de la(s) cámara(s) del dispositivo para tomar fotografías o para grabar vídeo.

12. Multimedia. Permiten reproducir y visualizar audio, vídeo e imágenes en el dispositivo.

- **Aplicaciones**

Es la capa superior, en la que se incluyen todas las aplicaciones del dispositivo, tanto las que tienen interfaz como las que no, las nativas (programadas en C o C++) y las administradas (programadas en Java), las que vienen preinstaladas en el dispositivo y aquellas que el usuario ha instalado.

En esta capa encontramos también la aplicación principal del sistema: Inicio (Home) o lanzador (launcher), porque es la que permite ejecutar otras aplicaciones mediante una lista y mostrando diferentes escritorios donde se pueden colocar accesos directos a aplicaciones o incluso widgets, que son también aplicaciones de esta capa.

❖ Características de la plataforma Android

Estas son las principales características y especificaciones actuales:

- Diseño del dispositivo
  La plataforma es adaptable a pantallas de mayor resolución, VGA, biblioteca de gráficos 2D y 3D (basada en OpenGL ES 2.0).
- Almacenamiento
  SQLite es la base de datos empleada en Android.
- Conectividad
  Android soporta tecnologías de conectividad como por ejemplo: GSM (EDGE, CDMA, Wi-Fi, LTE, NFC, Bluetooth, etc.
- Mensajería
  Además de los tradicionales SMS y MMS, la *Android Cloud to DeviceMessaging Framework* (C2DM) ya ha sido sustituida por la *Google Cloud Messaging (GCM)*como servicio *PushMessaging* de Android.
- Navegador web
  El navegador por defecto de *Ice CreamSandwich* (Android v4.0) obtiene ya una puntuación de 100/100 en el test Acid3 (sitio hecho por WaSP que pone a prueba los navegadores con los estándares web).
- Soporte de Java
  No hay máquina virtual Java, el código no se ejecuta sino que primero se compila en un ejecutable Dalvik y corre en la Máquina Virtual Dalvik.

- Soporte multimedia
  Android soporta multitud de formatos multimedia, tales como MP3, Ogg, WAV, PNG,…
- Soporte para streaming
- Soporte para hardware adicional
  Cámaras de fotos, de vídeo, acelerómetros, giroscopios, sensores de proximidad, presión y luz, etc.
- Entorno de desarrollo
  Eclipse es el más recomendado. Incluye herramientas de depuración de memoria, análisis del rendimiento del software y emulador de varios dispositivos. Sin embargo como emulador es más eficiente utilizar Genymotion o el propio dispositivo a través de la conexión con adb.
- Google Play
  Es el market de las aplicaciones, desde donde se descargan sin la necesidad de un PC.
- Multi-táctil
  Soporte nativo para pantallas capacitivas con soporte multi-táctil.
- Bluetooth
- Videollamada
  Desde HoneyComb (Android v3.0) tiene soporte para videollamadas a través de Google Talk.
- Multitarea
  Hace que las aplicaciones que no estén ejecutándose en primer plano reciban ciclos de reloj.
- Características basadas en voz
  Es posible abrir Google VoiceSearch y al decir “OK Google”, nuestro terminal recibirá órdenes como “Abrir x aplicación” entre otras opciones (no todas disponibles en todas las versiones Android).
- Tethering
  Es la característica que convierte a nuestro dispositivo en un punto de acceso inalámbrico y compartir así con otros terminales nuestra red.

❖ Aplicaciones de la domótica

- **Ahorro energético**

Mediante los sistemas de climatización y gestión eléctrica se puede conseguir un ahorro energético sin necesidad de la sustitución de aparatos ya instalados. Por ejemplo, quiero que se apague la calefacción a las 9 de la noche pero no estoy en casa, mando una orden a través de mi móvil para apagarla y que no se quede encendida hasta que llegue alguien a la vivienda

- **Confort**

Dentro del confort se pueden incluir todos los sistemas que mejoren la comodidad de una vivienda, como por ejemplo los sistemas de iluminación, integración del portero al teléfono o del videoportero al televisor, control vía internet, etc.

- **Seguridad**

Sistemas que proveen seguridad tanto a los bienes patrimoniales como a las personas. Son ejemplos las alarmas de intrusión, la teleasistencia, acceso a cámaras IP, etc.

- **Comunicaciones**

Son aquellas infraestructuras o dispositivos de comunicaciones que posee la vivienda. Ejemplos de ello son la ubicuidad en el control remoto desde Internet, mandos inalámbricos, intercomunicadores, informes de consumo y costes, entre otros.

- **Accesibilidad**

En general al utilizar la domótica en hogares y edificios lo que se está haciendo es que esos lugares sean accesibles para todos. Se adapta el entorno a todo tipo de persona sea cual sea su limitación o discapacidad, y con ello se ofrece más autonomía al individuo en sus tareas y quehaceres cotidianos.

Figura 64. Servicios domóticos

Las ventajas de la domótica forman una cadena que va desde el primer eslabón: facilitar la vida diaria a personas dependientes o con discapacidad hasta su relación con el exterior. Con los servicios tecnológicos integrados en su hogar se fomenta su comunicación con el exterior, se facilita la intercomunicación con familiares o asistentes, o con personal sanitario en caso de necesitarlo (teleasistencia).

En los sistemas domóticos debe primar su utilidad, usabilidad y flexibilidad, para que cada uno pueda utilizarlo según sus gustos, preferencias o necesidades. Un mismo sistema debería servir para que lo utilicen diversas personas, porque sería irreal o ilusorio crear mil conexiones distintas para mil perfiles de usuario.

Algunos ejemplos de aparatos que facilitan la vida diaria a personas con discapacidad:

- Teléfonos con sensores visuales y vibración para personas con discapacidad auditiva, y que a su vez, poseen teclas grandes y sonido para personas con discapacidad visual.
- Interfaces inalámbricos que permiten controlar aparatos solo con un movimiento de cabeza (FATRONIK)
- Productos para comunicar a través del iris (IRISCOM)
- Trajes robóticos para facilitar movimiento de extremidades

Para que los servicios de domótica sean accesibles y útiles para personas con discapacidad deben permitirles utilizar los aparatos tecnológicos y electrónicos básicos de toda casa (léase la televisión, su PC, los teléfonos, etc.) de forma fácil y sencilla; así como servirles de apoyo en su higiene diaria, al moverse o trasladarse por la casa, al comer, etc.

La idea es que puedan controlar toda la casa sin tener que realizar esfuerzos o desde el lugar donde se encuentren, lo cual se va a traducir en una serie de ventajas importantísimas para terminar con las barreras de accesibilidad:

- Mejorar la autonomía de las personas y fomentar su vida independiente
- Incrementa la calidad de vida y bienestar del usuario
- Mayor seguridad ante imprevistos (inundaciones, entrada de ladrones en la casa, etc.)
- Mejora la intercomunicación e integración tanto laboral, como social y emocional.
- Los cuidadores y asistentes también mejoran su calidad de vida, al verse apoyados en su tarea diaria con la ayuda de sistemas tecnológicos.
- Menor coste en contratar servicios asistenciales

ANEXO II: Estructura de tablas de Requisitos de Usuario

- **Nombre**: Nombre identificativo del requisito. Ha de ser unívoco y se utilizará a modo de resumen de dicho requisito. No tiene que incluir detalles, pero sí ser descriptivo.

- **ID**: identificador unívoco del requisito de usuario. Su nomenclatura ha de seguir el formato: UR_X_YY, donde:
  - UR significa requisito de usuario
  - X tomará los valores 'C' o 'R', según sea un requisito de capacidad o de restricción
  - YY será sustituido por el número de requisito dentro de su categoría, empezando por 01 e incrementando 1 en cada nuevo requisito.

- **Descripción**: Descripción detallada del requisito. Todos los detalles han de incluirse aquí para no dar lugar a ambigüedades futuras.

- **Prioridad**: Indica la prioridad de implementación del requisito. Facilita la organización y carga de trabajo a llevar durante el desarrollo del proyecto. Puede tomar los valores *Baja*, *Media* y *Alta*.

- **Necesidad**: Indica cómo de necesaria es la introducción del requisito en el proyecto. Puede tomar los valores *Esencial* y *Deseable.*

- **Estabilidad**: Indica si el requisito puede ser objeto de modificaciones durante el ciclo de vida del proyecto, o si por el contrario se trata de un requisito que no ha de sufrir variaciones. El campo puede tomar los valores *Estable* e *Inestable,* dependiendo de si se da el segundo o primer caso, respectivamente.

- **Verificabilidad**: Establece con qué facilidad se puede comprobar que el requisito haya sido introducido en el proyecto. Los valores que puede tomar son *Alta*, *Media* y *Baja*.

ANEXO III: Estructura de tablas de Requisitos de Software

- **Nombre:** Nombre identificativo del requisito. Ha de ser unívoco y se utilizará a modo de resumen de dicho requisito. No tiene que incluir detalles, pero sí ser descriptivo.

- **ID:** identificador unívoco del requisito de usuario. Su nomenclatura ha de seguir el formato: SR_X_YY, donde:
    - SR significa requisito de software
    - X será 'F' o 'NF', según sea un requisito funcional o no funcional
    - YY será sustituido por el número de requisito dentro de su categoría, empezando por 01 e incrementando 1 en cada nuevo requisito.

- **Descripción**: Descripción detallada del requisito. Todos los detalles han de incluirse aquí para no dar lugar a ambigüedades.

- **Prioridad**: Indica la prioridad de implementación del requisito. Facilita la organización y carga de trabajo a llevar durante el desarrollo del proyecto. Puede tomar los valores *Baja*, *Media* y *Alta*.

- **Necesidad**: Indica como de necesaria es la introducción del requisito en el proyecto. Puede tomar los valores *Esencial y Deseable.*

- **Estabilidad**: Indica si el requisito puede ser objeto de modificaciones durante el ciclo de vida del proyecto, o si por el contrario se trata de un requisito que no ha de sufrir variaciones. El campo puede tomar los valores *Estable e Inestable*, dependiendo de si se da el segundo o primer caso, respectivamente.

- **Verificabilidad**: Establece con qué facilidad se puede comprobar que el requisito haya sido introducido en el proyecto. Los valores que puede tomar son *Alta, Media y Baja.*

- **Dependencias UR**: Indica qué requisito de usuario se evalúa o implementa con dicho requisito software, puesto que cada requisito de usuario debe estar cubierto por uno o varios requisitos software.

ANEXO IV: Casos de uso particulares

Estos son los cuadros explicativos de los casos de uso derivados del caso de uso 3: Seleccionar acción a ejecutar. Tenemos 5:

- **Caso de Uso 4**

ESCANEAR	
Actor	Usuario de la aplicación.
Objetivo	Volver a ejecutar el reconocimiento de QR para obtener los datos pertenecientes al aparato elegido.
Pre-condición	- Conexión a internet - Conexión con el servidor que contacta con la BBDD - Enfocar la cámara hacia un código QR
Escenario	Usuario frente al aparato elegido para utilizar.
Post-condición	Se obtienen los datos de la QR y se envían al servidor para la obtención de la información almacenada en el mismo.

- **Caso de Uso 5**

CONTACTAR CON SOPORTE	
Actor	Usuario de la aplicación.
Objetivo	Contactar mediante un email con el soporte técnico del fabricante del electrodoméstico, para comunicar cualquier problema.
Pre-condición	- Conexión a internet - Tener configurada cuenta de correo en el móvil
Escenario	Usuario ordena la apertura de la interfaz de editar un email, lo redacta y lo envía.
Post-condición	Se envía el email y se devuelve el control a la aplicación *DomoScan*.

- **Caso de Uso 6**

EJECUTAR ACCIÓN	
Actor	Usuario de la aplicación.
Objetivo	Mandar una orden de ejecución al electrodoméstico elegido.
Pre-condición	- Tener activo el reconocimiento de voz.
Escenario	Usuario que selecciona una opción como por ejemplo abrir frigorífico, apagar lavadora, etc., mediante una orden por voz.
Post-condición	Se saca un mensaje en pantalla si se ha reconocido la orden correctamente, para en un futuro mandarla al software instalado en el electrodoméstico en cuestión.

- **Caso de Uso 7**

REPETIR OPCIONES	
Actor	Usuario de la aplicación.
Objetivo	Volver a escuchar las opciones disponibles.
Pre-condición	- Conexión a internet - Estar habilitado el reconocimiento de voz - Tener el dispositivo con volumen
Escenario	Usuario escuchando la reproducción tras ejecutar la acción por voz.
Post-condición	Se repite el audio con las opciones disponibles.

- **Caso de Uso 8**

CERRAR APLICACIÓN	
Actor	Usuario de la aplicación.
Objetivo	El cierre de la aplicación.
Pre-condición	- Tener habilitado el reconocimiento de voz.
Escenario	Usuario manda la orden por voz de cerrar aplicación.
Post-condición	Se cierra la aplicación.

ANEXO V: Manuales de Usuario y Administrador

Este anexo trata de aportar unos sencillos pasos para que usuario y administrador puedan manejar las utilidades atribuidas cada uno.

❖ **Información para el Usuario**

1. Para utilizar *DomoScan* debe en primer lugar abrir la aplicación, si su móvil lo soporta mediante una orden por voz.
2. Tras una pantalla con el logo y el título, aparece directamente la cámara para el reconocimiento de una etiqueta que tenga instalada en cualquier electrodoméstico.
3. Una vez reconocida sin que deba hacer nada, aparecerá en la pantalla un listado de opciones disponibles que será reproducido mediante un audio por si la información no se visualizara correctamente por la distancia al terminal.
4. Dichas opciones serán las propias del electrodoméstico seleccionado, cerrar la aplicación, realizar un nuevo escaneo o mandar un email al soporte técnico del fabricante.
5. Después se le solicitará una orden y usted deberá decirla en voz alta, tras lo cual la aplicación la llevará a cabo.

❖ **Información para el Administrador**

Entiéndase por administrador cualquier persona o entidad que provea de electrodomésticos al usuario de la aplicación. Como administrador tendrá dos opciones: o utilizar las credenciales de Quickblox para almacenar el contenido o bien montar su propio servidor. Con la primera tan sólo deberá solicitar al desarrollador las credenciales para poder acceder al servicio en la nube y subir ahí el contenido necesario definido en el apartado 3.2.3 de la memoria. Si en cambio el administrador prefiere tener su propio backend, que sería lo más recomendable, deberá tener en cuenta varios principios básicos:

- Que la QR contenga la URL comenzando por http:// y conteniendo los parámetros inequívocos de la búsqueda.
- El servidor debe aceptar y responder a peticiones post que son las que están implementadas en la aplicación.
- La respuesta deberá ser devuelta en formato JSON para que *DomoScan* sepa interpretarla.

Bibliografía

1. [En línea] http://www.quees.info/que-es-un-smartphone.html.

2. [En línea] http://es.kioskea.net/faq/8470-analisis-de-los-sistemas-operativos-para-smartphones.

3. [En línea] http://androideity.com/2011/07/04/arquitectura-de-android/.

4. [En línea] http://www.androidcurso.com/index.php/tutoriales-android/37-unidad-6-multimedia-y-ciclo-de-vida/158-ciclo-de-vida-de-una-aplicacion.

5. [En línea] https://play.google.com/store.

6. [En línea] http://www.cedom.es/sobre-domotica/publicaciones.

7. [En línea] http://es.wikipedia.org/wiki/Dom%C3%B3tica.

8. [En línea] http://www.whatisaqrcode.co.uk/.

9. [En línea] http://www.mediaq.es/aplicaciones-codigo-qr/tamanos-de-los-codigos-qr.html.

10. [En línea] http://zbar.sourceforge.net/.

11. [En línea] https://github.com/zxing/zxing.

12. [En línea] https://support.google.com/websearch/answer/2940021?hl=es.

13. [En línea] http://gerenciaydesarollodesoftware.blogspot.com.es/2013/03/matriz-de-trazabilidad.html.

14. [En línea] http://cooltext.com/Logos.

15. [En línea] https://sites.google.com/site/aprendeappinventor/documentacion/activity-starter.

16. [En línea] http://admsaludv.wordpress.com/59-2/.

17. [En línea] http://www.slideshare.net/DACB_Lcc/dispositivos-moviles-1639336.

Glosario

CONCEPTO	DESCRIPCIÓN
PDA	Personal Digital Assistant
PC	Personal Computer
App	Aplicación
S.O.	Sistema Operativo
TTS	TestToSpech, que es texto para reproducir en un audio
MVC	Modelo Vista Controlador
BBDD	Base de datos
DuPont	Empresa dedicada a varias ramas industriales
T38	Protocolo de Fax
UML	Modelo lenguaje unificado
API	Application Programming Interface, conjunto de llamadas que ofrecen acceso a funciones y procedimientos representando una capa de abstracción para el desarrollador
Librería	Agrupación de código que proporcionan servicios a programas independientes pasando a formar parte de éstos. Permiten la distribución de funcionalidades y la construcción modular de un proyecto
XML	eXtensible Markup Language o lenguaje de marcas extensible, metalenguaje que permite definir la gramática de otros lenguajes específicos
Kernel	Núcleo de un S.O. y por lo tanto pieza clave para el funcionamiento del mismo. Es responsable de dar acceso al hardware, gestionar recursos y hacer llamadas al sistema
Interfaz	Concepto referido a la abstracción que un determinado elemento o conjunto de elementos realiza sobre sí mismo, facilitando el uso y acceso por otros elementos externos
Framework	Término con el que se define un amplio conjunto de elementos que permite el desarrollo y organización de software utilizando un determinado lenguaje o tecnología
Bytecode	Código intermedio, más abstracto que el código máquina, y que necesita un mediador o máquina virtual para poder ser transformado en código nativo y ejecutado por el hardware
Tap	Término utilizado para referirse al acto de interactuar con la aplicación a través de ejecutar una acción en la pantalla
URL	Uniform resource locator, localizador uniforme de recursos
ADB	Android Debug Bridge, permite lanzar una aplicación Android desde el ordenador a un Smartphone en modo debug
APK	Application Package File, es un paquete para el S.O. Android

Universidad Carlos III de Madrid

RESUMEN EXTENDIDO DEL TRABAJO FIN DE GRADO

DISEÑO E INTEGRACIÓN EN ANDROID DE UN SISTEMA DE DOMÓTICA ASISTENCIAL BASADO EN RECONOCIMIENTO DE IMÁGENES

Alumno: Paloma Diego Velasco Tutor: Javier Fernández Muñoz

PLANTEAMIENTO DEL PROBLEMA

Las barreras arquitectónicas que existen aún hoy para las personas con cualquier tipo de discapacidad motriz son más que evidentes. Si pensamos en ello, en seguida se nos viene a la cabeza, estaciones de metro sin ascensor, acceso sin rampa a portales, supermercados, bares u otros establecimientos; pero además de estas situaciones, ¿qué pasa con las más cotidianas? Aquellas que desempeña la persona en su propia casa, tales como encender la televisión, abrir una ventana, encender un horno o abrir el frigorífico, parecen quizá menos problema pero cómo hacer éstas tareas sin movilidad en el cuerpo y sin depender de otra persona.

De este planteamiento nace la necesidad de crear herramientas que puedan facilitar en la mayor medida posible el día a día de una persona. El planteamiento básico es diseñar algo que pueda permitir al usuario interactuar con los aparatos de su casa, en concreto nos centraremos en electrodomésticos, con una premisa fundamental: deberá poder realizar cualquier acción sin necesidad de tocar, manipular o mover nada, para adaptar el sistema a la inmovilidad del usuario sentado en una silla de ruedas. Es el punto en que llegamos directamente al término de domótica, concepto que básicamente consiste en crear un hogar que pueda ser controlado por el usuario de manera remota, desde un mando, una orden por voz o una secuencia de comandos programado.

Y ahora, ¿cómo interactuar con un electrodoméstico? El escenario que planteamos para la resolución de esta problemática es la siguiente:

- Se colocará una etiqueta en cada aparato sobre el cual se actuará, que identificará el electrodoméstico en cuestión.
- El usuario se encontrará sentado en su silla de ruedas frente al electrodoméstico elegido.
- En la silla habrá instalado un soporte que mantendrá un Smartphone.
- El usuario abrirá la aplicación desarrollada para este proyecto.

Desde este momento el usuario tendrá el control total sobre el electrodoméstico de igual modo que si pudiera caminar, pudiendo realizar cualquier acción con tan sólo solicitarla mediante una orden por voz al móvil, el cual o bien directamente ejecutará o enviará al aparato para ser ejecutada.

Por lo tanto el siguiente paso será decidir cómo solucionar el problema planteado, que por lo que ya se intuye vendrá de la mano de una aplicación móvil para Smartphone, cuyo desarrollo tendrá que cumplir además los siguientes sub-objetivos:

- ✓ Reconocer los aparatos utilizando etiquetas (distancia usuario-etiqueta de 1.5m mín.)
- ✓ Almacenar en BBDD los datos referentes al electrodoméstico.
- ✓ Creación de una cadena de audio que reproduzca al usuario las opciones disponibles.
- ✓ Reconocimiento de voz para captar la opción que desea el usuario.
- ✓ Hacer una estructura de proyecto enfocada a que pueda ser la base de una línea futura más compleja.
- ✓ Realizar pruebas que verifiquen el funcionamiento de la aplicación.
- ✓ Adaptar la aplicación a posibles cambios descubiertos en la fase de test que puedan mejorar la experiencia de usuario.
- ✓ Hacer la aplicación funcional tanto de la manera tradicional interactuando con la pantalla como sin necesidad de tocarla, y siempre de un modo muy sencillo.
- ✓ Desarrollar el código de la aplicación de una manera clara, eficiente y estructurada.

SOLUCIÓN TÉCNICA AL PROBLEMA

Básicamente, ya hemos visto que el proyecto implementará una aplicación llamada *DomoScan* que proveerá al usuario de un punto de independencia en su vida. Para llevarla a cabo se emplean las siguientes tecnologías debidamente justificadas:

- ❖ **SMARTPHONE Y ANDROID**

Puesto que la aplicación a desarrollar será utilizada desde una silla de ruedas, el dispositivo que la alberque deberá ser un dispositivo móvil que pueda trasladarse con la silla de un lado para otro. Planteada esta cuestión, surgen dos disyuntivas:

- *Qué tipo de dispositivo móvil elegir*

Si consultamos en internet el número de usuarios de Smartphone frente a los usuarios con un teléfono tradicional las cifras son aplastantes, por lo que pensar en una aplicación para el "teléfono inteligente" es una buena idea si intentamos llegar al mayor número de usuarios que esté en nuestra mano. Además se elije directamente un Smartphone y no otras posibilidades similares como una tablet, por su reducido tamaño, ya que la portabilidad del dispositivo es fundamental a la hora de que el usuario se mueva de un lado a otro por la casa.

- *Sobre qué plataforma trabajar*

Son muchos y variados los S.O. que han ido saliendo para los móviles, incluso hace relativamente poco, nació uno nuevo propio de la compañía Mozilla Foundation, llamado Firefox OS, y es de prever que sigan saliendo más. Sin embargo, hoy por hoy Android es el dueño del mercado común, así como en las altas esferas el uso de iOS está más extendido, por lo general el usuario de a pie opta por un Smartphone con Android, ya que su precio es mucho más asequible y la calidad de funcionamiento es bastante buena.

Otro motivo fundamental en la elección de esta plataforma es el hecho de que sea una plataforma libre, lo que se traduce entre otras características en que el programador no pagará nada por desarrollar para Android. Dadas estas características Android es la plataforma elegida. El modelo del que se dispone para la emulación del código y las pruebas finales es el Samsung Galaxy S.

➔ Para la elaboración del proyecto el entorno de desarrollo utilizado será ECLIPSE, por sus innumerables ventajas entre las que se encuentra una perfecta integración con la plataforma Android a través de la descarga del ADT, emulador del código, servidor adb para emulación en el propio terminal, etc.

❖ **ETIQUETAS QR**

Se crearán etiquetas QR en una web destinada a este fin de manera libre y se instalarán en electrodomésticos para su lectura. Como ejemplo de QR podemos ver la imagen que tenemos a la derecha de estas líneas. Vemos que es un código bidimensional, por lo que almacenará información en horizontal y vertical. Los QR se clasifican según el número de información que almacenan, lo cual va relacionado con el número de columnas que tengan, así un QR v1 será construido con 21filas x 21columnas.

Mucho se puede decir de estos códigos y su reconocimiento, pero sobre todo destaquemos en este resumen lo que los diferencia de otros: que contienen información por sí mismos, (datos alfanuméricos) y que son fácilmente reconocidos.

❖ SERVICIO EN LA NUBE: QUICKBLOX

Tras plantearse qué información debía contener la QR, si toda la información relativa al electrodoméstico, incluyendo o no la posibles acciones a realizar, o sólo un id que lo relacionara con una BBDD SQLite en local, se llegó a la conclusión de que había que generalizar lo máximo el proyecto para que cualquiera pudiera usarlo siguiendo unas sencillas pautas. Por ello se elige la opción de que la QR contenga la marca y un id único que coincidan con los almacenados en una BBDD remota con la que nos comunicaremos a través de un servidor. Puesto que el objetivo de este proyecto no era implementar un backend, se utiliza un servicio en la nube llamado Quickblox que nos proveerá de todo el backend preciso, haciendo uso de su librería y de la propia interfaz web.

❖ LIBRERÍAS UTILIZADAS

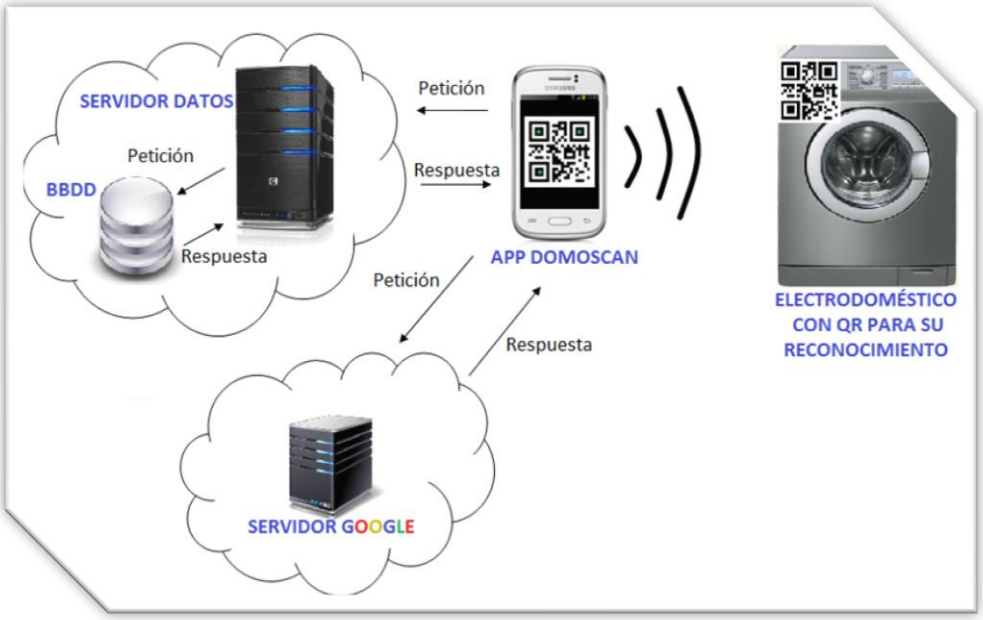
Para la implementación del proyecto se utilizarán 3 librerías:

- zXing para el reconocimiento de QRs,

- quickblox-android-0.1.6, librería utilizada para la gestión de la información en el servicio web utilizado, que es Quickblox.

- android-async-http-1.4.4 para gestionar peticiones post a cualquier servidor. El motivo de este desarrollo es estandarizar más aún el uso del sistema creado por el lado de los fabricantes, los cuales podrán crear sus QRs con la petición a un servidor propio, siguiendo unas pautas marcadas por el desarrollador (detalladas en el anexo Manual de Administrador)

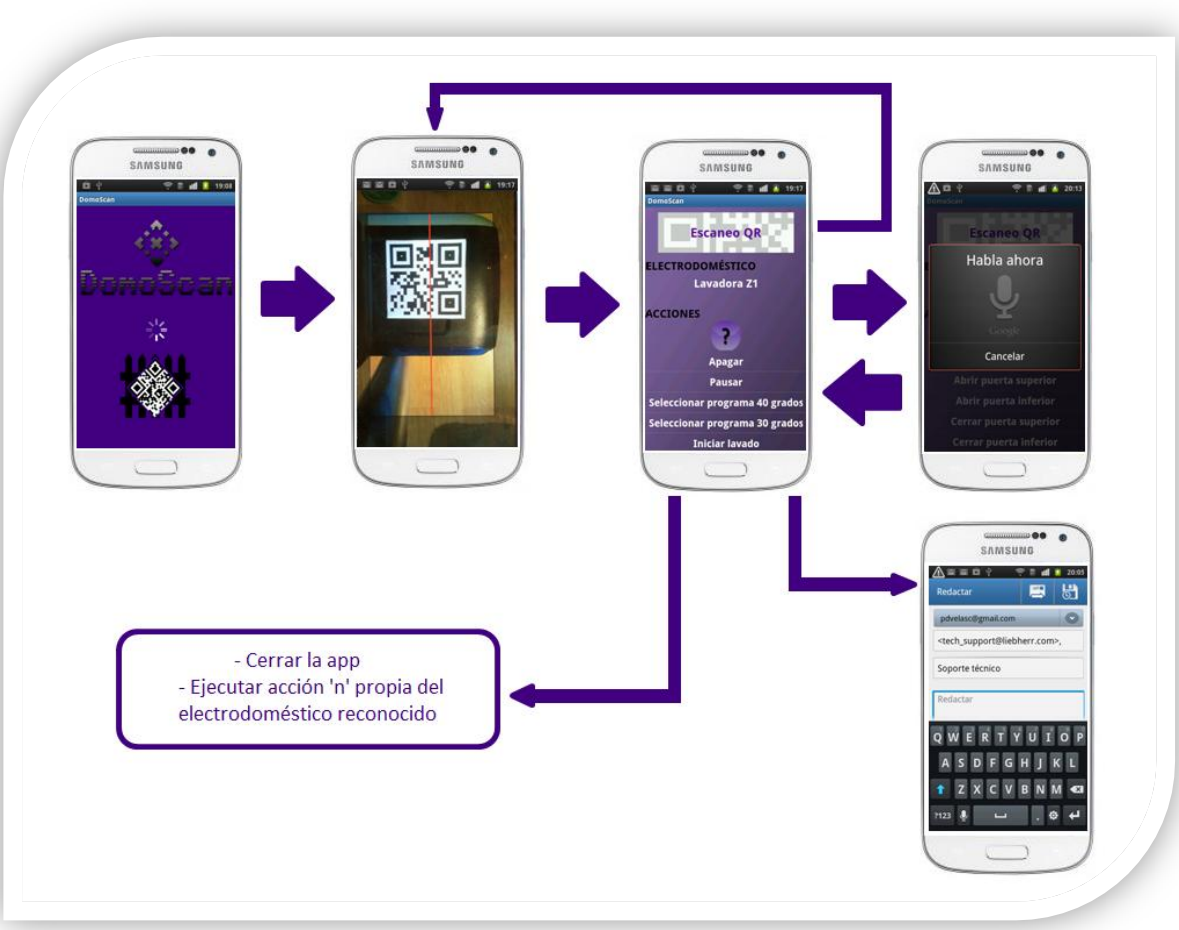
Utilizando todas estas tecnologías y entrelazándolas entre sí llegamos a tener la siguiente arquitectura del sistema completo:



Arquitectura del sistema

RESULTADOS Y CONCLUSIONES

Tras la implementación de la aplicación que cumpliera los requisitos descritos, llamada *DomoScan,* se llevaron a cabo las correspondientes pruebas que verificaran el correcto funcionamiento. Tal y como demuestran las matrices de trazabilidad expuestas en la memoria de este proyecto, los objetivos fueron cumplidos y el resultado es una aplicación sencilla de manejar con el menor número de operaciones por parte del usuario. La navegación consiste en una pantalla inicial en la que se cargan datos y que desaparecerá tras concluir su función (lo que generalmente se conoce como Splash), tras la cual se mostrará directamente la cámara para realizar el reconocimiento de una etiqueta para posteriormente llevar al usuario a la pantalla principal donde se muestran los datos recopilados del servidor para el electrodoméstico reconocido, además de otros botones como el de volver a escanear un código o contactar con el servicio técnico. Vamos a ver el resultado con las pantallas y la navegación entre ellas:



Pantallas de la aplicación DomoScan y su navegabilidad

Como vemos en la imagen, tras el reconocimiento aparece la pantalla principal con la información recopilada del servidor. En ese momento lo que no se aprecia en la figura es el audio que se reproduce tras cargar todos los datos provenientes del servidor. Dicho audio enumera las opciones disponibles para que el usuario las conozca sin necesidad de ver la pantalla, por si la localización del Smartphone en la silla no permitiera la lectura. Acto seguido a la escucha se inicia el reconocimiento de voz, momento en el que el usuario podrá dar una orden, la cual será reconocida por Google Voice quien enviará la respuesta a la aplicación. Posteriormente la opción seleccionada desencadenará un suceso como los descritos en la imagen anterior. En el caso de seleccionar una acción propia del electrodoméstico se mostrará un mensaje en la pantalla puesto que la aplicación queda pendiente de la integración con un sistema domótico a desarrollar.

Una vez concluido el desarrollo del proyecto, es momento de hacer balance y ver si los objetivos planteados han sido logrados.

La finalidad buscada por este proyecto era implementar una aplicación Android que reconociera etiquetas situadas en electrodomésticos y se pudiera manejar a través de órdenes por voz, para poder ser utilizada por personas con una discapacidad física grave en la movilidad. A la vista de los resultados el funcionamiento satisface los requisitos y cumple con los objetivos marcados (probado en profundidad en la matriz de trazabilidad con las pruebas funcionales realizadas).

En referencia a la implementación del código ha habido dos aspectos claves para llevar a cabo fácilmente modificaciones futuras y en gran medida para el aprendizaje del alumno responsable del desarrollo.

En primer lugar, se ha programado utilizando el MVC y aprovechando también la separación interfaz-lógica que ya de por sí tiene la programación en Android hacen que cualquier persona pueda continuar desarrollando modificaciones futuras. Se pensó así desde el principio ya que el proyecto deberá integrarse en un futuro con un sistema domótico y las acciones no sólo mostrarán mensajes en la pantalla sino que deberá desarrollarse código que mande las órdenes desde el Smartphone hasta el electrodoméstico.

Por otro lado, la conclusión personal viene marcada por el aprendizaje que se ha obtenido en la realización del proyecto. Un objetivo muy claro era ser desarrolladora de aplicaciones en el futuro, por lo que quería aprovechar el proyecto para programar de un modo más eficaz y organizado. He aprendido entre otras cosas a utilizar patrones de diseño, programar dividiendo clases según funcionalidades, diseñar adapters personalizados en Android, etc., métodos que hoy utilizo habitualmente en el trabajo y cuyas características aprendí con la realización de este proyecto.

LÍNEA FUTURA

En un primer momento el desarrollo de *DomoScan* y su integración con un sistema de domótica asistencial real pueden ser muy beneficiosos para mejorar la calidad de vida de los usuarios. Sin embargo una línea futura basada en la utilización de las Google Glass, puede ser una realidad muy provechosa y revolucionaria para aportar independencia a las personas con movilidad reducida.

Las Google Glass son unas gafas que ha desarrollado Google que llevan cámara de 5MP de resolución, un touchpad en el costado que permite a los usuarios controlar el dispositivo mediante gestos como desplazar o tocar, tiene todas las características principales de *Smartphone* como giroscopio, acelerómetro, Bluetooth, conexión Wi-Fi, etc., posee también sensores de luz ambiente y de proximidad, un sistema de inducción ósea para la transmisión del sonido, entre otras muchas características.

Examinando las cualidades que tienen las Google Glass, podemos pensar en un sinfín de posibilidades, entre las cuales pueden encontrarse algunas muy importantes que supondrían un cambio radical en la vida de muchas personas con variados tipos de discapacidad, entre las que podemos enmarcar a los usuarios considerados para el desarrollo de este proyecto. Incluso se comenta que los cristales podrán ser graduados, por lo que mejor aún será la usabilidad para las personas con discapacidad, algunas de las cuales tienen problemas en la movilidad derivados de una enfermedad neurológica que en muchas ocasiones afecta al habla o la visión, por lo que el hecho de que las gafas puedan ser graduadas y manejarse con movimientos de cabeza facilitará el control de las aplicaciones.

Considerando este aspecto, señalar que la aplicación *DomoScan* ha sido desarrollada en Android y no en otra plataforma, en gran medida pensando en la idea de adaptar el código a las Google Glass, ya que esta tecnología tiene un software basado en la versión de Android 4.0.4, por lo que en principio no debería ser muy costoso la adaptación del código fuente para que la aplicación funcione en las gafas.

Aunque aún sólo están disponibles para programadores residentes en EEUU (para los usuarios estuvieron a la venta sólo el 15 de abril del presente año) por el precio de 1500$, Google ya ha expresado en foros su intención de sacarlas en España antes del término del 2014 con un valor aproximado de 400€. Realidad o no, lo cierto es que en algún momento las gafas estarán disponibles para usuarios y desarrolladores de todo el mundo, momento en el que espero que esta aplicación y otras similares puedan ser desarrolladas y adaptadas para su funcionamiento en las Google Glass.